TREBALL FINAL DE GRAU

TÍTOL: Multi Intelligent Agent Unit per a una Smartcity

AUTORS: PEDRAZA SANTOS, ADRIÁN

DATA DE PRESENTACIÓ: Octubre, 2019

COGNOMS: PEDRAZA SANTOS **NOM: ADRIÁN**

TITULACIÓ: Grau en Enginyeria Informàtica

PLA:

DIRECTOR:

DEPARTAMENT: Arquitectura de computadors

QUALIFICACIÓ DEL TFG

TRIBUNAL

PRESIDENT **SECRETARI** **VOCAL**

DATA DE LECTURA:

Aquest Projecte té en compte aspectes mediambientals: □ Sí □ No

Resum

Actualment les ciutats són els nuclis urbans on més grups de personen conviuen. Encara i que les ciutats són una font de riquesa també requereixen de múltiples recursos per al seu manteniment i correcte desenvolupament de la vida de les persones que hi resideixen.

Davant de la necessitat d'optimitzar els recursos disponibles i degut als avenços tecnològics a nivell de xarxes, dispositius i maquinària han sigut molt notables va aparèixer un nou concepte de ciutats, les anomenades ciutats intel·ligents.

Les ciutats intel·ligents o Smartcities són aquelles ciutats que compten amb diversos dispositius que recullen tot tipus d'informació d'aquesta. El que determina com d'intel·ligent és una ciutat és la plataforma que l'administra. Aquesta plataforma s'encarrega d'utilitzar els recursos disponibles de la manera més eficient possible a partir de la informació proporcionada per aquests dispositius. Degut a la novetat d'aquest concepte actualment hi ha diverses línies d'investigació que busquen com desenvolupar aquesta plataforma de la millor manera possible.

L'objectiu principal d'aquest projecte és dissenyar i desenvolupar una plataforma que permeti gestionar i utilitzar els recursos d'una Smartcity. Aquesta plataforma és construïda a partir de la connexió i comunicació de diferents tipus de màquines entre si. En aquest projecte s'ha dissenyat i construït un prototip d'un node capaç de participar en la gestió d'una Smartcity.

Paraules clau: plataforma ,recursos , dispositius, Smart city.

Abstract

Currently, the cities are the main place where groups of people come together. Even though the cities are a source of wealth that also requires multiple resources for the maintenance and correct development of the lives of the people who reside there.

In front of the need to optimize the available resources and because of the technological avenues at networking, devices and machinery have been remarkable a new concept has appeared, the smart cities.

The Smartcities are those cities that have various devices that gather all kind of information from it. What determines how intelligent a Smartcity is the platform that administers it. This platform is responsible for using the resources available in the most efficient way possible based on the information provided by these devices. Due to the novelty of this concept, there are currently several lines of research that seek to develop this platform in the best possible way.

The main objective of this project is to design and develop a platform that allows to manage and use the resources of a Smartcity. This platform is built from the connection and communication of different types of machines to each other. In this project, a prototype of a node capable of participating in the management of a Smartcity has been designed and constructed.

Key words: platform, devices, resources, SmartCity.

Índex general

Índex d'imatges

Índex de taules

Glossari

- Agent: qualsevol tipus de màquina que pugui formar part de la plataforma d'administració d'una Smartcity. El terme node s'utilitza de manera sinònima.

- API: Application Programming Interface o interfície de programació d'aplicacions és el conjunt de mètodes que una aplicació permet executar des d'aplicacions externes.

- Device: el terme que s'ha utilitzat per fer referència al dispositiu (no confondre amb IoT) dissenyat i construït en aquest projecte. Aquesta màquina porta incorporat el software de la aplicació desenvolupada.

- Front-end: pantalla que permet la interacció i visualització d'informació per part d'un usuari.

- IoT: Internet of Things o Internet de les coses és el concepte que fa referència a la comunicació de tot tipus de dispositius a través d'una xarxa. En aquest projecte s'utilitza per referir-se a aquells dispositius intel·ligents com barreres, sensors de tot tipus... El terme dispositiu és utilitzat de manera sinònima a aquest.

- Raspberry pi: màquina de mesures reduïdes i de baix cost. Aquests dispositius compten amb el seu sistema operatiu propi anomenat Raspbian.

- Servei: funcionalitat o aplicació pràctica que es porta a terme en una Smartcity degut a la plataforma que l'administra.

- Sockets: mecanisme de comunicació que permet la comunicació entre dues aplicacions diferents. Cada socket bé definit per la direcció ip, el protocol de transport i un número de port.

- Topologia: terme utilitzat per fer referència al conjunt d'elements de la ciutat que participen d'alguna manera en la gestió d'una Smartcity. Estructura i plataforma s'utilitzen també per fer referència a aquest terme .

- Zona: àrea de la ciutat on es troba un agent leader. Aquest leader pot enregistrar nous agents que es trobin en aquesta zona.

1.INTRODUCCIÓ

1.1 Context

Les millores tecnològiques dels últims anys han desembocat en que cada persona disposa d'un petit **ecosistema** de dispositius que li permet realitzar tot tipus de tasques més còmodament. Per exemple, els smarthomes d'una casa permeten el control de la seguretat, la programació de tasques, el control de llum i de temperatura... Els wearables ens permeten monitoritzar la nostra salut, els smartphones ens permeten controlar altres dispositius...

Avui en dia succeeix el mateix en algunes ciutats, on s'han incorporat dispositius que tracten de captar informació del tràfic, d'humitat, de temperatura... La tendència del mercat[1] i les necessitats de la nostra societat indiquen que finalment comptarem amb un gran ecosistema de dispositius en les nostres ciutats. Aquests dispositius ens permetran amb l'administració adequada fer la nostra ciutat més intel·ligent a partir de la informació proporcionada.

Una ciutat intel·ligent o **Smartcity** [2] és aquella ciutat que compta dispositius que capten tot tipus d'informació per efectuar un correcte ús dels recursos disponibles (personal humà , d'infraestructures, de maquinària) de la ciutat. El principal objectiu d'una Smartcity és millorar la vida dels ciutadans com a resultat d'un correcte ús de la informació disponible.

Per tal d'assolir aquest objectiu una Smartcity disposa d'una **plataforma** o estructura (o com anomenem en el projecte, topologia) encarregada de gestionar i utilitzar tota la informació rebuda dels dispositius dispersos per la ciutat. Tradicionalment aquesta plataforma està dividida en dos nivells: un primer nivell format per dispositius físics que capten dades constantment i un altre nivell que es troba al "núvol" on s'emmagatzemen i tracten aquestes dades. Aquesta estructura és coneguda com el model tradicional basat en IoT i Cloud Computing.

El model de basat en Cloud Computing utilitza la informació emmagatzemada en centres de dades que es troben a fora de la ciutat (normalment en altres països).

Com que l'ús d'aquesta informació per nodes de la ciutat no és tan àgil pel volum de dades i la distància s'utilitzen dades lleugerament desfasades al moment actual.

Com que l'administració de la ciutat té com a requeriment una màxima eficiència operacional, les màquines encarregades de la computació es trobaran en la mateixa ciutat. Aquest nou nivell s'anomena "**fog**" o boira i s'encarrega de realitzar una gestió dels recursos a partir de la informació que es disposa en temps real. Aquesta capa com a principal avantatja respecte el model anterior és que permet la interacció en temps real amb els dispositius disponibles.

En aquest projecte s'ha desenvolupat la plataforma d'Smartcity amb el model Fog-To-Cloud, és a dir, amb la inclusió d'aquest nivell d'intel·ligència. Aquest nivell està format per nodes o **agents** [9] que s'encarreguen d'administrar la ciutat en temps real, així com d'efectuar l'execució de serveis. Aquest model també compta amb la capa d'IoT i el Cloud, encara que aquesta última passa a tenir menys responsabilitats. Addicionalment, aquesta plataforma ha de poder ser monitoritzada remotament.

Els **serveis** d'una ciutat intel·ligent són aquelles aplicacions o eines que s'ofereixen al ciutadà com a resultat de la plataforma d'una Smartcity. Un exemple de servei podria ser el canvi d'estat de semàfors o de barreres per una ambulància que porta un pacient o buscar l'aparcament més proper segons el tràfic de la ciutat en el moment de la petició.

Aquests agents podran ser tot tipus de màquines dins de la ciutat: des de dispositius mòbils i cotxes autònoms fins a portàtils/ordinadors sobre taula. Una de les finalitats d'aquest projecte és el **disseny i construcció** d'un dispositiu capaç d'executar-se com a agent.

Aquesta plataforma podrà ser monitoritzada des d'una màquina externa. En aquesta màquina es trobarà el **front-end**, on es mostra l'estat de la plataforma. Des de quins nodes estan disponibles en el moment actual com quines tasques estan portant a terme.

1.2 Origen del projecte

Aquest projecte és una proposta del grup d'investigació multidisciplinari CRAAX (Centre d'Arquitectures Avançades de Xarxes) de Vilanova i la Geltrú on es va fer latent la necessitat de disposar d'una estructura que s'encarregués de gestionar els diversos recursos d'una ciutat intel·ligent.

Com que no es disposa d'una ciutat intel·ligent com a tal (amb milers de dispositius diferents dispersos en una zona) en aquest grup d'investigació es va construir un escenari de proves o **testbed** que simula una Smartcity ja que compta amb automòbils, semàfors, il·luminació pròpia...

En aquest centre d'investigació es va arribar a la conclusió que calia algun tipus d'administració sobre els recursos d'una Smartcity. A partir d'aquesta necessitat va sortir la idea de realitzar aquest projecte.

1.3 Objectius

Com s'ha mencionat anteriorment aquest projecte té com a principal objectiu el disseny i la implementació d'una plataforma intel·ligent per a una Smartcity.

Aquesta plataforma està pensada per ser establerta mitjançant una aplicació que permet a qualsevol màquina que la pugui executar ser participant de l'administració d'una ciutat intel·ligent. Un altre dels objectius principals d'aquest projecte és dissenyar aquesta aplicació de la millor forma possible.

Com a últim objectiu del projecte es pretén dissenyar i construir un prototip per a una màquina que pugui ser participant de la gestió d'una Smartcity.

1.3.1 Objectius tècnics

Els objectius tècnics són aquells que es pretenen acomplir a nivell tècnic o professional durant el transcurs de l'acompliment dels objectius principals. Els objectius tècnics d'aquest projecte són:

- Dissenyar i implementar una plataforma utilitzant les tecnologies apropiades.
- El disseny i implementació d'un agent.
- La implementació de diversos serveis que permetin testejar el funcionament de la plataforma.
- El disseny i implementació d'un front-end per a monitoritzar cadascun dels nodes que formen part de la plataforma.
- La implementació de l'aplicació usada per l'usuari client de la plataforma.
- El redactat de la memòria tècnica del projecte per tal de que pugui ser utilitzada per altres desenvolupadors.

1.3.2 Objectius acadèmics

Els objectius acadèmics són aquells que s'estableixen per a l'aprenentatge durant el transcurs del projecte. Els objectius acadèmics d'aquest projecte són:

- Utilitzar l'aprenentatge obtingut en la universitat per portar a terme el projecte mitjançant una metodologia de treball.
- Col·laborar conjuntament amb estudiants del mateix grau que estan realitzant projectes relacionats amb l'àmbit de les ciutats intel·ligents.
- L'assoliment dels objectius plantejats inicialment per tal d'aportar el resultat del projecte al grup CRAAX.
- Assentar les bases d'una futura línia de treball en l'àmbit de l'administració i gestió d'una ciutat intel·ligent.

1.4 Continguts

El projecte es troba dividit en els següents apartats:

- 1. **Introducció**: en aquest primer apartat es descriu quins han sigut els motius que han donat lloc a la realització d'aquest projecte i quins són els principals objectius a assolir.

- 2. **Metodologia**: apartat on s'explica la metodologia de treball que s'ha seguit per dur a terme aquest projecte.
- 3. **Estat de l'art**: introducció a l'àmbit sobre el qual es basa aquest projecte: que és una Smartcity i com està formada, quines són les investigacions actuals en aquest àmbit... Finalment es descriu el model d'Smartcity amb el que es basa el projecte.
- 4. **Anàlisi de requisits:** apartat on s'analitzen els requisits funcionals del projecte, esmentant les restriccions d'aquest.
- 5. **Disseny**: en aquest apartat es detalla com s'han dissenyat la plataforma, els agents, els serveis , el front-end i l'aplicació client.
- 6. **Desenvolupament tècnic**: en aquest apartat es realitza una explicació tècnica de la posada en pràctica del disseny.
- 7. **Testeig del projecte**: apartat on es mostren els resultats obtinguts per les proves que ens permeten comprovar el funcionament de la implementació.
- 8. **Integració amb altres projectes:** apartat on s'explica com s'ha desenvolupat aquest projecte per poder integrar el resultat amb projectes del mateix àmbit realitzats per altres desenvolupadors.
- 9. **Device**: en aquest apartat es mostra el procés el disseny i construcció d'un dispositiu capaç d'executar el software d'un agent.
- 10. **Projectes futurs:** capítol on es plantegen els principals punts a desenvolupar com a continuació del projecte.
- 11. **Conclusions:** apartat on es descriuen punt per punt les conclusions a les que s'ha arribat amb aquest projecte, realitzant una comparació amb els objectius inicials.
- 12. **Pressupost del projecte**: apartat on es detalla el cost del projecte si aquest es posés en venta.
- 13. **Bibliografia**: apartat on es llisten les fonts bibliogràfiques consultades per a la realització del projecte.
- 14. **Referències**: apartat on es llisten les referències del projecte.
- 15. **Annexos**: annexos del projecte. Un d'ells tracta sobre l'estudi de llibreries en el llenguatge de programació python i en l'altra es descriuen els manuals d'instal·lació i configuració de màquines i de software per a reproduir aquest projecte.

2. METODOLOGIA

Aquest projecte s'ha desenvolupat seguint una metodologia de treball determinada, per poder acomplir els objectius proposats en l'apartat anterior de la millor forma possible.

2.1 Metodologia de desenvolupament

Aquest projecte ha sigut realitzat amb una metodologia que ha sigut adaptada de dues metodologies àgils conegudes: l'Scrum i el Kanban .

La metodologia Scrum és la metodologia on es realitzen entregues "petites" d'un increment en el projecte cada 2 setmanes aproximadament. De tal manera que si en un moment s'ha de modificar alguna part ja desenvolupada solament aquella part sigui la que es modifiqui. Cadascun dels períodes de treball del projecte s'anomena **sprint.**

La metodologia Kanban permet tenir una visió molt global del projecte a partir de les tasques que s'han de fer, s'estan fent i les que estan fetes.

2.1.1 Seguiment

El seguiment d'aquest projecte s'ha realitzat amb l'aplicació **Trello**. Aquesta aplicació permet tenir tant a professor com alumne una visió global del projecte: quins són els objectius a curt i llarg termini i quines tasques s'han realitzat o queden per finalitzar.

La divisió de les columnes és pròpia dels que usen el taulell, però en general trobem les següents columnes en aquells projectes que usen metodologies àgils:

- **Èpiques**: les èpiques del projecte són els grans objectius/tasques a realitzar que no es poden resoldre en un sprint. Les èpiques d'un projecte es defineixen al principi d'aquest, per tal de tenir una visió global del projecte.
- **Històries d'usuari** o Sprint Backlog: són aquells objectius que es poden resoldre en un sprint. A l'inici de cada sprint es defineixen les històries

d'usuari segons l'objectiu a assolir. La resolució de les històries d'usuari permeten assolir les èpiques plantejades inicialment.

- **Parking:** en aquest apartat es col·loquen aquelles tasques que s'han de realitzar en algun moment del projecte però que no són de caràcter urgent per les prioritats del projecte. Típicament es col·loquen aquelles tasques d'anotacions o de revisions del desenvolupament.
- **To do**: tasques que queden per realitzar.
- **Doing**: tasques que s'estan portant a terme en el moment actual.
- **Done**: tasques completades fins al moment. Una tasca es dóna per acabada quan s'han obtingut els resultats esperats (això implica que s'ha realitzat un testeig exhaustiu per arribar a la solució esperada).

Quan un sprint finalitza s'avaluen les tasques que han quedat per fer (si hi ha), es defineixen les noves històries d'usuari i es col·loquen les noves tasques a realitzar.

Gràcies a aquesta aplicació conjuntament amb les demostracions de final d'sprint (realitzades en powerpoint) es manté en tot moment un historial dels increments realitzats i presentats en el projecte.

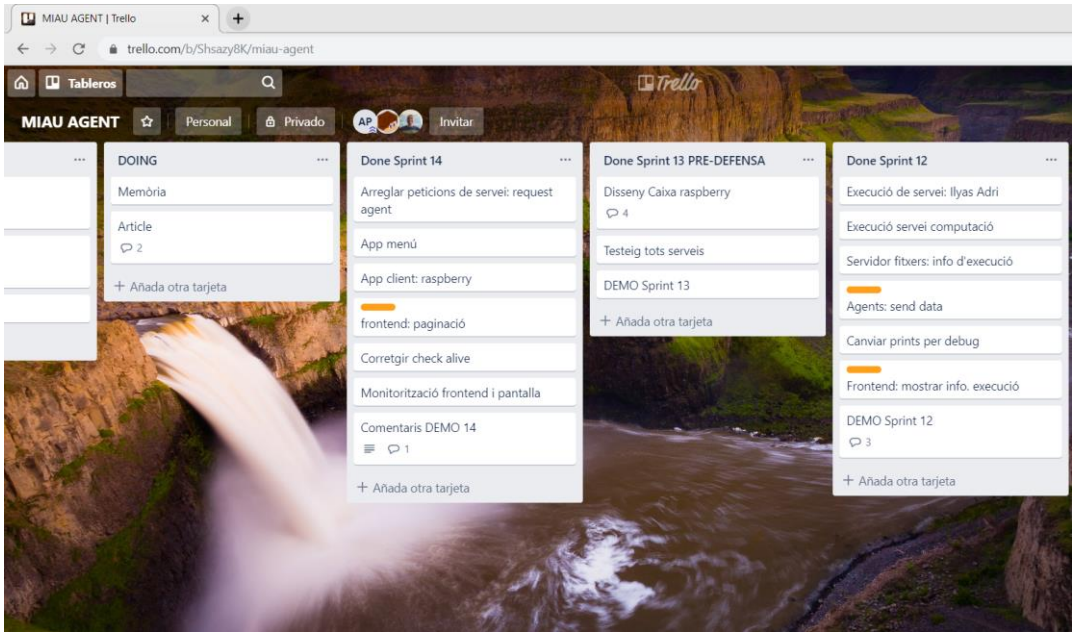


Fig. 1: Versió web de l'aplicació Trello.

2.2 Reunions

Com s'ha comentat anteriorment les reunions entre professorat i alumne s'han realitzat aproximadament cada dues setmanes. S'ha anomenat **demo** a la presentació del producte o dels avenços fets al final d'un sprint. Aquestes demos es basaven en:

- Un resum del que s'havia fet en l'sprint anterior, per tal de situar professor i l'alumne.
- Sprint actual: en aquest apartat s'explicaven els diferents avenços fets en el projecte durant l'sprint. Aquests avenços es demostraven realitzant una petita demostració del funcionament dels avenços fets.
- Pròxim sprint: a partir dels avenços fets es proposaven els pròxims punts del projecte que es resoldrien en el següent sprint.

A vegades, conjuntament amb els avenços es realitzava una **proposta** o idea per tal de resoldre algun a dificultat/problema que havia succeït durant el transcurs de l'sprint. Aquesta proposta es feia al professor degut a que el punt en qüestió podia ser resolt de maneres diferents, cadascuna amb les seves avantatges i inconveniències.

2.3 Resum temporal del projecte

En la taula 1 es pot veure el resum temporal del projecte:

Resum del projecte	
Data d'inici	29/11/2018
Data de fi	13/10/2019
Data de lectura	22/10/2019
Duració dels sprints	~2 setmanes
Dedicació semanal	18 hores
Hores totals previstes	540 hores
Hores finals dedicades	Aprox.. 560 hores

Taula 1: Resum temporal del projecte.

2.4 Resum temporals dels sprints

En la taula 2 es pot veure el tema principal de cada sprint així com les dates d'inici i final:

Sprints	Data inici	Data fi
Sprint 0: Introducció	29/11/2018	13/12/2018
Sprint 1: Primera versió d'agent	14/12/2018	16/1/2019
Sprint 2: Raspberry pi	17/1/2019	30/2/2019
Sprint 3: Canals de comunicació	31/1/2019	13/2/2019
Sprint 4: Servei canvi de color	14/2/2019	27/2/2019
Sprint 5: Enregistrament d'agents i cloud	28/2/2019	13/3/2019
Sprint 6: Detecció de caigudes	14/3/2019	27/3/2019
Sprint 7: Canvi de rols	28/3/2019	10/4/2019
Sprint 8: Pla de dades	11/4/2019	2/5/2019
Sprint 9: Comunicació entre agents	3/5/2019	15/5/2019
Sprint 10: Servei Cloud	16/5/2019	29/5/2019
Sprint 11: API i catàleg de serveis	30/5/2019	1/7/2019
Sprint 12: Integració del projecte i monitorització	2/7/2019	11/7/2019
Sprint 13: Finalització del projecte	12/7/2019	13/10/2019

Taula 2: Resum temporal dels sprints.

2.5 Descripció dels sprints

A continuació s'explica en termes generals el progrés fet en cada sprint del projecte.

2.5.1 Sprint 0 Introducció

En aquest sprint inicial es va realitzar una investigació profunda sobre les smartcities. Què són i com estan formades, quina estructura tenen, com afecten als ciutadans...per tal de determinar quin model de gestió d'Smartcity és el més adequat per implementar. Aquest sprint també va servir per definir els principals objectius del projecte.

En aquest sprint no es va realitzar una entrega del producte com a tal, sinó que aquí es va realitzar el redactat del apartat del projecte Estat de l'art.

2.5.2 Sprint 1 primera versió d'agent

En aquest sprint es va desenvolupar una primera versió del que són els agents en el nostre model d'Smartcity escollit anteriorment. El producte a entregar consistia en comunicar diverses màquines mitjançant un canal bidireccional de comunicació. Amb aquest producte es va desenvolupar una aproximació de comunicació bàsica entre diversos agents.

2.5.3 Sprint 2 : Raspberry pi

En aquest sprint es va realitzar una primer disseny 3D del dispositiu físic utilitzant com a dispositiu una raspberry pi. En aquest sprint es va posar en marxa la raspberry pi utilitzada. A més, es va realitzar una investigació de protocols i llibreries per tal de comunicar els agents entre si de la millor manera possible, per tal de veure si hi havia alternatives a la proposada en l'sprint anterior que poguessin oferir millores en quant a la comunicació.

2.5.4 Sprint 3: Canals de comunicació

En aquest sprint es va plantejar un nou model d'agent que permetia la comunicació entre agents per diferents canals de comunicació, segons la finalitat del missatge. En aquest sprint es van identificar i llistar les funcionalitats prioritàries a desenvolupar i es van identificar aquells mòduls necessaris per l'agent físic. Addicionalment, es va realitzar la instal·lació del servidor Cloud (escollit en l'sprint anterior).

2.5.5 Sprint 4: Millora del agent i servei canvi de color

En aquest sprint es va finalitzar la implementació del model d'agent presentat en l'sprint 3 que permetia la comunicació entre agents per diferents canals. Per comprovar el seu funcionament, es va realitzar una primera versió del servei canvi de color de semàfor. Addicionalment, es va realitzar una proposta del sistema de caigudes: la notificació d'estats entre agent i leader.

2.5.6 Sprint 5: Enregistrament d'agents i Cloud

En aquest sprint es aconseguir assolir una comunicació entre el servidor i un agent, per tal de que aquest pugui emmagatzemar informació a partir del registre. A més, es va definir una primera estructura de Base de dades per emmagatzemar informació de dispositius i d'agents. Per finalitzar, es va realitzar una millora

important en la manera en com s'agreguen agents a la plataforma.

2.5.7 Sprint 6: Detecció de caigudes

En aquest sprint es va desenvolupar gran part del sistema que permetia prevenir la caiguda d'agents de l'estructura. Es va realitzar una detecció de caigudes que permet identificar quan un agent o leader es troben caigut.. Juntament amb la detecció es van desenvolupar els procediments i el sistema que permeten escollir un nou leader de la zona. Addicionalment, es va proposar una primera versió modular del agent.

2.5.8 Sprint 7: canvi de rols

Relacionat amb el treball desenvolupat en l'sprint anterior en aquest sprint es va desenvolupar el sistema que permet que un agent canviï de rol (passar d'agent a leader) per tal d'assegurar el funcionament de l'estructura.
En aquest sprint es va realitzar la primera aproximació amb l'API i la Base de Dades topològica basada en MongoDB. Addicionalment, es va realitzar una proposta de resolució de servei basada en la delegació de responsabilitats.

2.5.9 Sprint 8: Pla de dades

En aquest sprint es va realitzar un altre canvi en l'enregistrament d'agents i dispositius. En aquest punt del projecte es va fer canvis importants a nivell de codi en diferents aspectes per utilitzar elements que s'havien començat a desenvolupar anteriorment: com l'ús de les Bases de Dades locals i la BBDD topològica basada en MongoDB . A causa d'aquest canvi es va realitzar una primer aproximació al que seria el disseny modular del software dissenyat. Finalment, es va començar a implementar el servei canvi de color semàfor mitjançant la resolució del cloud utilitzant la seva base de dades.

2.5.10 Sprint 9: Comunicació entre agents

En aquest sprint es va realitzar el primer disseny 3D amb el mòdul PoE incorporat a la raspberry pi. A més, es van incloure mètodes de validació d'agents per part dels leaders. En aquest sprint es va finalitzar exitosament un servei que involucrava la comunicació entre agents.

2.5.11 Sprint 10: Servei cloud

En aquest sprint es van realitzar diferents millores en diversos apartats. Primerament es va establir la comunicació directa entre leaders i cloud, mitjançant un enregistrament. Es va millorar l'execució de serveis: amb la implementació d'un mètode d'actualització de paràmetres i la finalització del servei (tancament de connexions). Per concloure també es va finalitzar el disseny del prototip 3D del agent físic.

2.5.12 Sprint 11: API i catàleg de serveis

Aquest sprint es divideix en dues weeklys (reunions setmanals) després de la qual es va realitzar una demo "final" del projecte.

Primerament es va desenvolupar una millor API per la comunicació entre agents. Es van desenvolupar els canvis necessaris en altres mòduls del projecte per la seva correcta execució. També es va adaptar els serveis implementats fins al moment a la incorporació del catàleg de serveis i l'API.

En aquest sprint es va desenvolupar la primera versió del front-end que permetia visualitzar la informació d'agents i serveis.

2.5.13 Sprint 12 Integració del projecte i monitorització

Aquest sprint es va modificar el disseny de l'agent per tal d'aconseguir una integració amb projectes del mateix àmbit de les Smartcities.

En aquest sprint també es va dur a terme la monitorització dels agents de la topologia a partir de l'enviament de dades de cadascun d'ells. El front-end es va adaptar per mostrar correctament la informació enviada pels agents.

2.5.14 Sprint 13: Finalització del projecte

En aquest sprint es va realitzar un testeig general del projecte mitjançant la realització de diverses proves.

Addicionalment s'ha intentat millorar el disseny (a nivell de codi) dels diferents mòduls. Com a conseqüència el front-end es va actualitzar per tal de mostrar tota la informació correctament: tant la dels serveis (dels serveis externs al projecte) com de la base de dades topològica.

3. ESTAT DE L'ART

Actualment les ciutats són els llocs on més persones es congreguen per a desenvolupar la seva vida ja sigui per viure, treballar o viatjar. Les Smartcities o **ciutats intel·ligents** és un concepte que va sorgir com a conseqüència dels avenços tecnològics i davant la necessitat de millorar en tots els àmbits possibles la qualitat de vida de les persones. Aquests avenços tecnològics s'han donat a nivell de xarxes, de dispositius (els anomenats **IoT**) i de maquinària en general.

Degut a la novetat del concepte , les Smartcities es troben en el punt de mira de desenvolupament de diversos investigadors. Encara i que hi ha diferències en com implementar aquest concepte.

A continuació es detallen els conceptes esmentats, així com els principal models de ciutats intel·ligents.

3.1 Smartcities

Una ciutat intel·ligent o **Smartcity** [2] en anglès és tota aqueta ciutat que integra i monitoritza les seves infraestructures (carreteres, aeroports, comunicacions, energia...) per optimitzar millor els **recursos[1] disponibles.** Per a la realització de plans de manteniment, monitorització i seguretat, sempre amb l'objectiu de proporcionar serveis funcionals pels ciutadans.

En definitiva, una ciutat intel·ligent promociona la inversió en capital humà i en infraestructures de Tecnologies de la Informació i la Comunicació (TIC) per aconseguir una millora de la qualitat de vida dels ciutadans. Aquesta estructura estarà dividida entre les màquines que s'encarreguen de l'administració i els dispositius que recullen informació.

[1] Un recurs d'una ciutat intel·ligent és tot allò que intervé i s'utilitza en una ciutat (per exemple: personal humà, servei d'aigua, institucions, infraestructures...)

La infraestructura física agregada que es basa en l'Internet de les Coses (IoT). Tots els models de ciutats intel·ligents dissenyats fins ara tenen en comú que en la infraestructura agregada siguin aquests els responsables de recollir algun tipus d'informació i transmetre-la de forma eficient.

La infraestructura lògica intel·ligent de la nostra Smartcity serà una **plataforma** o estructura de nodes encarregada de la gestió de recursos. Aquesta estructura o plataforma té l'objectiu de l'administració i aprofitament de recursos que disposa una ciutat i la reducció de costos. Aquesta plataforma actuarà segons la informació rebuda dels IoT.

El producte resultant d'una estructura lògica ben dissenyada d'una Smartcity són els **serveis**. Els serveis són aplicacions complexes dirigides i ofertes als ciutadans, que utilitzen la informació generada d'una ciutat intel·ligent. Encara que existeixen serveis dirigits al propi manteniment de la ciutat. Aquestes aplicacions poden ser creades pels responsables de la plataforma o per empreses externes que vulguin participar. Els tipus de serveis que podem trobar en una Smartcity són els següents:

Fig. 2: Principals àrees de desenvolupament d'una Smartcity.

3.1.1 Plataforma intel·ligent

La **infraestructura lògica** d'una Smartcity és la que ens determina en gran mesura com d'intel·ligent és una Smartcity. En general una ciutat serà considerada com a intel·ligent quan:

- Es disposa d'un sistema administrador intel·ligent de tota la ciutat. Això implica la correcta gestió dels recursos, la monitorització de l'estat dels dispositius i dels diferents components al servei de la ciutat (Policials, hospitalaris, manteniment...) . Avui en dia també és molt important assegurar un nivell de ciberseguretat [3] del nostre servei TIC.
- Els **serveis** oferts als ciutadans són útils i solucionen alguna problemàtica d'aquests. Per exemple, que un pacient sigui atès a la menor brevetat de temps possible, que en la ciutat cada vehicle estigui seguint una ruta que li permeti arribar aviat al seu destí evitant la congestió en les carreteres, el manteniment immediat d'infraestructures com carreteres, ponts, edificacions públiques...

Aquests dos punts són en general, la columna vertebral d'una Smartcity a grans trets. Encara que el concepte de les ciutats intel·ligents sigui molt innovador i és el futur de les nostres ciutats podem observar com tots els models d'Smartcities dissenyats fins el moment tenen una sèrie de contrapartides o punts a millorar/tenir en compte:

- Per poder construir una ciutat intel·ligent cal disposar inicialment d'una **forta inversió** en sensors, plataformes (serveis TIC), xarxes, potència de computació, personal humà capacitat... per tal de garantir un funcionament òptim. La posada en marxa d'aquesta topologia és costosa i difícil de mantenir.

- Una ciutat que disposa de sensors que constantment estan en marxa transmeten informació, una computació que s'encarrega d'organitzar i monitoritzar tots els recursos d'una ciutat, un processament de dades, una comunicació de dades amb serveis exteriors... tot això implica un **cost energètic** que es suma al cost energètic d'una ciutat. Degut a això sempre

s'hauria de realitzar un estudi previ de les ciutats abans d'implementar una Smartcity. Doncs pot ocorre que en ciutats petites no sigui rentable (junt amb el punt anterior)aportar aquesta intel·ligència addicional.

- En un entorn tan summament gran on tenim molts components és important garantir un nivell de **ciberseguretat**[2] i d'**anonimat de les dades** extretes (ja que també s'extreuen de persones). Avui en dia la preocupació per com es gestionen la informació privada és latent. La plataforma ha d'estar dissenyada per evitar filtracions d'informació, de serveis, venta de dades... La possessió d'un volum tant gran d'informació sempre és objectiu de tercers per a usos fora de l'entorn d'una ciutat intel·ligent. [3]

- No solament s'ha de tenir en compte els canvis a realitzar en una ciutat per incorporar una **nova infraestructura** de dispositius, sinó també la implementació d'una correcta estructura dels diferents components TIC d'una Smartcity.

L'**estructura** d'una Smartcity no és una contrapartida sinó un punt que està en constant canvi avui en dia. No es pot trobar una fórmula que funcioni per a tots els casos (cada ciutat és única: disposa d'uns recursos concrets, té una localització determinada..). En casos reals d'avui en dia aquesta estructura és dissenyada per ser el més flexible i escalable possible a canvis d'entorn.

La millor solució sempre és mirar d'implementar una estructura dels diferents components que redueixi al mínim els temps de comunicació entre dispositius, que minimitzi el nombre de recursos a utilitzar per a un pla d'acció, que utilitzi models predictius[3] de dades útils i funcionals, que tingui un bon disseny que permeti l'accés de dades per àmbits o sectors de manera separada i conjunta... En definitiva el disseny d'un plataforma TIC determina com d'intel·ligent és una Smartcity. [4]

[2] Àmbit de la informàtica que s'enfoca en la protecció de la infraestructura computacional d'un entorn.
[3] Models de dades que utilitzen dades anteriors i noves per intentar predir resultats posteriors, avui en dia aquest camp de la informàtica s'està utilitzant en molts àmbits diferents.

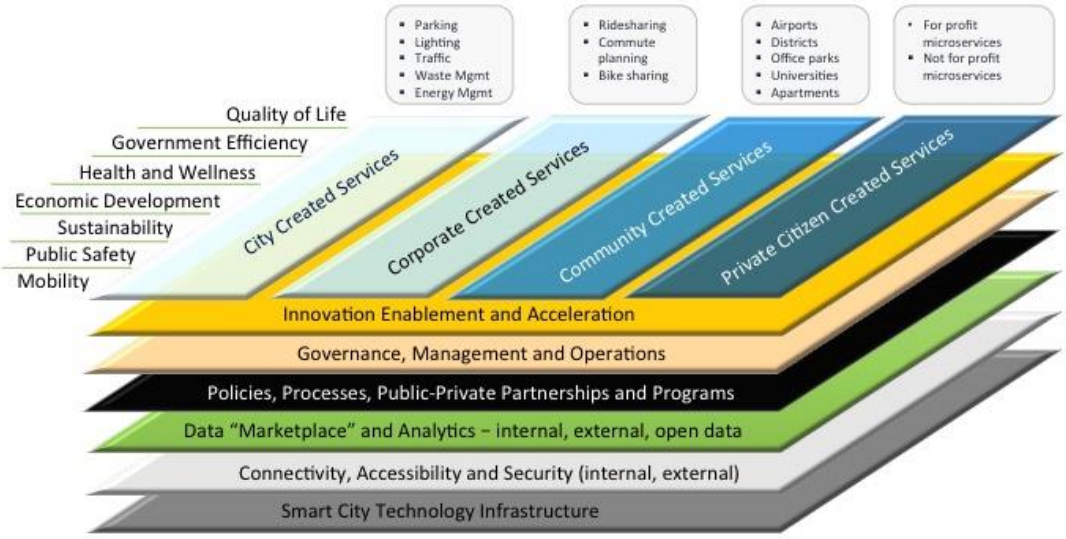


Fig. 3: Esquema de conceptes d'una Smartcity.

La imatge 3 representa el conjunt de conceptes a tenir en compte en el disseny d'una Smartcity. Els nivells inferiors que s'observen en la imatge representen el **conjunt de processos** invisible a un usuari consumidor d'un servei ofert per a una Smartcity. Sempre es parteix de la infraestructura formada per dispositius.

Posteriorment a l'ús de les dades es defineixen una sèrie de polítiques, aliances a nivell empresarial (es pot donar el cas on dues empreses diferents que utilitzen les mateixes fonts d'informació per a usos diferents). A continuació es realitza una gestió de de l'enviament dels resultats als consumidors (empreses, ciutadans, institucions públiques...). Aquest procés finalitza amb la innovació per part d'empreses i particulars, en forma de servei orientat al públic.

Podem distingir a l'esquerra els àmbits més importants que trobem en una Smartcity i a sobre de les capes abans mencionades els 4 tipus de serveis oferts, amb exemples del seu ús(en la part superior).

3.1.2 Internet of Things

L'internet de les coses o Internet of Things (IoT) és el concepte que fa referència a la comunicació entre diversos tipus d'objectes. Aquests objectes o **dispositius** poden ser remotament monitoritzats i configurats. Avui en dia els IoT són utilitzats en sistemes de control i automatització d'actuadors, en la domòtica, en sensors intel·ligents... Qualsevol dispositiu que compti amb un hardware i software adient pot ser un IoT o dispositiu intel·ligent.

El principal punt fort d'aquest concepte és la comunicació que s'estableix entre objectes diferents per assolir algun objectiu en comú. Un conjunt d'IoT resulten en un ecosistema que permet a l'usuari la realització de diverses tasques. Un exemple molt clar d'aquest fet és el concepte d'Smarthome.

En una ciutat intel·ligent aquests dispositius IoT tenen un funcionament diferent. Aquests estan programats per captar dades i enviar-les periòdicament a una capa superior de processat de dades. En aquest cas, no es realitzarà una connexió entre els dispositius IoT que estan dispersos per la ciutat.

Aquests dispositius tenen incorporats **sensors** que capten una informació del ambient on es troben (temperatura, humitat, número de persones en una habitació, quantitat d'un producte...) . Aquesta informació s'envia periòdicament a través de la xarxa a un node conegut. S'estableix una comunicació entre IoT i nodes que permeten la execució de serveis amb informació captada en temps real.[5]

Un **cas pràctic** de l'ús de la informació enviada d'un dispositiu IoT és el següent: imaginem un cas on tenim un grup de dispositius que compten el número de persones que es troben en les diferents sales d'un hospital . En un entorn de ciutat intel·ligent, quan un nou pacient es detecta l'ambulància encarregada de portar-lo aniria al hospital que estès més a prop i que pogués atendre millor al nostre pacient (per exemple si aquest necessités una operació quirúrgica, s'aniria al hospital més proper que tingués la sala d'operacions buida). Com que aquesta informació recollida pels sensors és actual i s'actualitza periòdicament es considera fiable. Aquesta informació ajuda a l'hora de traçar el **pla d'acció** adient, en aquest cas ha

sigut trobar l'hospital més adequat per un nou pacient.

Fig. 4: Els dispositius IoT es consideren proveidors d'informació en una Smartcity.

3.2 Cloud computing

Molts estudis en aquests últims anys apunten a que en un futur proper tindrem molt més dispositius connectats[1], així com persones que els utilitzen. A part dels sensors que composen la nostra columna vertebral d'una Smartcity.

Aquest conjunt de fets deixen clar que fa falta un servei encarregat únicament de l'**emmagatzematge i ús de dades** degut al volum tant gran d'informació resultant. Això és el que coneixem com computació al núvol o cloud computing, és la computació que es realitza a través de la xarxa de servidors, on les aplicacions s'executen a distància i la informació d'aquestes es guarda en centres de dades.
En una Smartcity un servei cloud s'encarrega de processar i emmagatzemar totes les dades que els diferents sensors que tenim recullen. El proveïdor d'aquest servei té la funció d'administrar tots els seus nodes per tal de garantir a una ciutat intel·ligent que la informació guardada és d'accessibilitat ràpida, multi plataforma i segura (que mai es perdrà informació).[6]

En definitiva un bon servei de Cloud ens ha d'assegurar els següents punts:

- Un **processament àgil** de les dades, per efectuar operacions de serveis més ràpidament.

- Un correcte **emmagatzematge de les dades** rebudes pel client. Això implica l'ús de backups[4] per evitar pèrdues d'informació i memòria suficient per al gran volum d'informació.
- **Accés ràpid i multi plataforma** de les dades per tal d'assegurar una bona velocitat de transmissió de dades amb el client.
- L'execució ràpida de **serveis** que s'executen en servidors Cloud (fig. 5). Aquests també s'encarreguen de garantir la qualitat d'aquestes aplicacions (l'emmagatzematge de dades dels usuaris, manteniment de les aplicacions...).

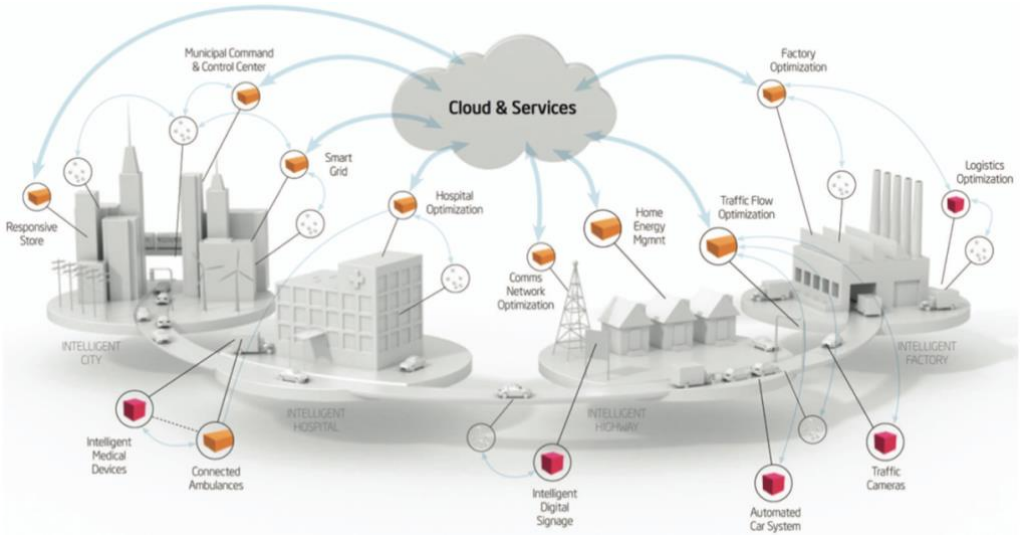


Fig. 5: Imatge d'una Smartcity basada en l'execució de serveis al Cloud.

3.3 Model tradicional d'una Smartcity

Avui dia s'ha estandarditzat el model de **Cloud Computing** o computació en el núvol i l'ús de dispositius **IoT** en el disseny d'una Smartcity. Per saber com funciona un model d'una ciutat intel·ligent cal analitzar la **successió de processos** que ocorren de principi a final d'aquest.

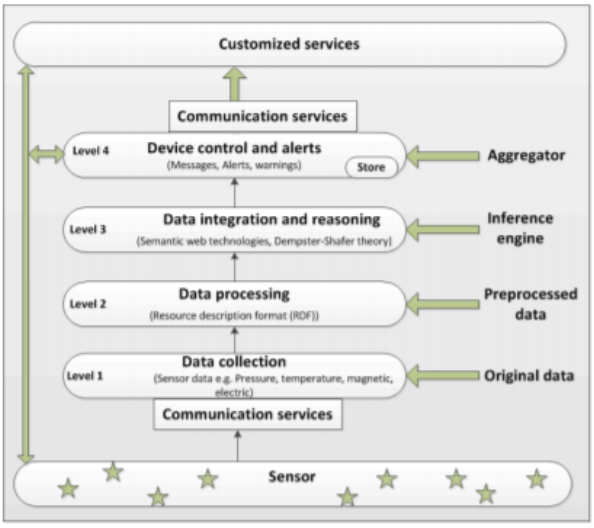
El punt de partida és un dispositiu IoT qualsevol que capta unes dades d'algun sensor que s' envien a un servidor cloud. Aquest servidor processa aquestes dades

[4] "Còpia de seguretat":conjunt d'arxius que es copia en un de sol per a la recuperació dels arxius originals en cas de que s'eliminin o es perdin.

(totes les dades enviades per tots els dispositius IoT que té al seu càrrec) per tal de poder-les tractar. Aquest processament s'encarrega d'adaptar les dades de formats diferents en un únic format per a facilitar el seu ús.

S'assumeix que aquests servidors estan dividits per àrees i que en cada àrea tenim diversos tipus de dispositius. L'ús de les dades bé determinat pel servei.

Aquestes dades utilitzades poden esdevenir en una comunicació amb altres components de la capa Cloud (emmagatzemar dades, intercanvi de missatges, sistemes d'alerta...). O bé l'ús de dades esdevé d'un **pla d'acció** que correspon a un servei o aplicació. Aquests dos casos donats són molt diferents degut a que en el primer s'ha de realitzar una comunicació amb components de la ciutat intel·ligent, per exemple amb finalitats de manteniment d'infraestructures. En canvi en el segon cas, l'ús d'aquestes dades tenen la finalitat d'utilitzar-se en un servei. La següent imatge resum els passos mencionats:[7]



Nivell 1: Recopilació de dades a través de dispositius IoT
Nivell 2: Processament de dades
Nivell 3: Integració de dades
Nivell 4: Control de dispositius.

Fig. 6: Esquema del procés de les dades d'una Smartcity.

3.3.1 Problemes d'aquest model

Aquest model és el més estandarditzat avui en dia. La capa IoT s'encarrega de proporcionar informació en temps real i la capa Cloud de la seva computació i ús. Encara que aquest model presenta un problema greu, l'excessiva càrrega de responsabilitats de la capa Cloud, sent la capa on s'efectuen la majoria de tasques.

Els nivells 2,3 i 4 (fig. 6) són efectuats per màquines que utilitzen la computació al núvol i això repercuteix en els temps de comunicació entre nivells.
El processat de dades s'hauria de realitzar per màquines que estiguin a prop dels dispositius IoT. És a dir, en la mateixa ciutat o voltants per millorar l'eficiència en general. Si no es pot reduir el volum de dades, s'intenta minimitzar els temps de transmissió degut a la distància. Aquest punt podria ser resolt si es disposés de xarxes 5G degut a la baixa latència d'aquestes.

El punt més feble és el nivell d'Integració de dades degut a la distància entre nodes. Per realitzar models predictius de dades s'utilitzen tant les dades emmagatzemades com les noves dades agregades. Com que les dades anteriors estan emmagatzemades en el núvol (podem assumir que en servidors que es troben a altres països) no es pot realitzar una integració ràpida de dades. Una solució a aquest problema seria disposar d'una part del emmagatzematge en la pròpia ciutat i cada cert temps informar a un servidor del núvol on ja es dipositen dades que s'utilitzaran amb menys freqüència.

Per últim trobem al problema del nivell de control de dispositius. Aquest nivell solament l'acompleixen aquells dispositius que estan al servei municipal: per exemple per tasques de manteniment, o d'alerta. És a dir, podem trobar casos on després de realitzar una integració de dades podem passar directament al servei final. Per exemple: una aplicació meteorològica que a partir de les temperatures actuals i anteriors (dels passats dies per exemple) prediu la temperatura més probable de les pròximes hores o dies. En aquest cas no ens faria falta el nivell 4 de control de dispositius i d'enviament de senyals perquè aquesta informació es podria transmetre directament als responsables de l'aplicació, Això ens genera una bifurcació al passar del nivell 3. [8]

Per tant , en alguns casos hauríem de disposar de components capaços de decidir que es fa amb les dades processades: o bé s'avisa a altres dispositius o bé es transmet la informació directament al responsable d'un servei.

3.5.Model Fog to Cloud

En aquests últims anys s'ha estandarditzat el cloud computing per a les ciutats intel·ligents però com em vist anteriorment aquest té algunes contrapartides. Aquest model afegeix és l'addició d'una nova capa anomenada "fog" o "boira" situada entre la capa IoT i el Cloud com a solució de les problemàtiques del model anterior.

La **computació Fog** és aquella arquitectura que ens facilita les operacions de computació, emmagatzematge i serveis de xarxa entre dispositius i la computació Cloud, a través d'una xarxa en un nivell més proper a l'usuari, en la pròpia ciutat.
El model **Fog-to-Cloud** o MF2C té com a principal objectiu dissenyar i desenvolupar una plataforma d'administració segura, oberta, jeràrquica i descentralitzada per facilitar l'eficient ús de recursos, tenint en compte els requeriments dels serveis i la demanda del mercat combinant la computació Cloud i Fog . (*Towards an Open, Secure, Decentralized and Coordinated Fog-to-Cloud Management Ecosystem (mF2C),* pàgina 2).

Descripció breu de les característiques del model MF2C:

- **Oberta:** és una capa dissenyada per tal de que el públic (empreses de desenvolupament informàtiques , professionals autònoms) pugui incorporar funcionalitats i aplicacions, experimentar i crear.
- **Segura:** com passava en la computació cloud, aquest model ha de garantir un fort nivell de ciberseguretat en cadascun dels nivells (Iot, Fog i Cloud) que la composen.
- **Descentralitzada:** aquest model és tolerant a fallades dels seus components, com que això és possible la informació es guarda en diversos punts, per si un d'aquests cau evitant així dades perdudes.
- **Coordinada:** aquest model està dissenyat per resoldre les diferents tasques i serveis mitjançant la interacció entre components.

La funció principal que té aquesta capa és dur a terme aquelles tasques que es poden realitzar millor dins de la pròpia ciutat. Aquesta metodologia permet obtenir millors resultats, guanyant rapidesa en les operacions reduint temps de comunicació innecessaris.

Per dissenyar un MF2C funcional designarem un nou conjunt de responsables encarregats de portar a terme les funcions de la capa Fog, els **agents**.[9]

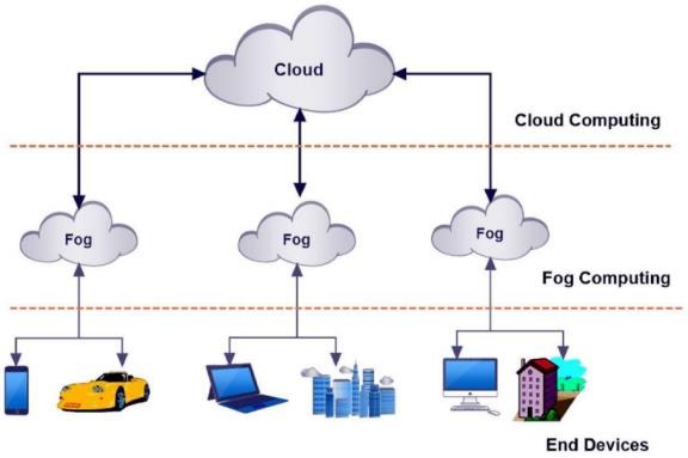


Fig. 7: Esquema del model Fog-to-Cloud.

3.5.1 Agent

Un **agent** [9] és aquell node que pertany a la capa fog dins del nostre model MF2C que es situa entre els dispositius IoT i el cloud. És una màquina qualsevol (ordinador, raspberry, telèfon...) que executa un software(una aplicació) que el fa participant de la plataforma de gestió d'una Smartcity.

En el nostre model tindrem diferents agents que es comuniquen entre si per dur a terme una millor execució de les tasques. Aquesta capa divideix una tasca gran en petites on aquestes són executades paral·lelament per a finalitzar les tasques més ràpidament.

Per definir la sèrie de funcionalitats de les que disposa un agent caldrà observar model tradicional degut a que l'agent passarà a tenir responsabilitats que abans tenia el cloud. Els agents de la ciutat comptaran amb les següents funcionalitats:

- **Comunicació amb els dispositius IoT:** l'agent ha d'estar preparat per rebre dades d'aquests dispositius.
- **Processament de les dades** que es reben per part de molts dispositius IoT que envien dades en diferents formats. Aquestes dades se'ls hi dóna un mateix format per poder-les utilitzar per igual. (figura 6 nivell 2).

- **Gestió de les dades:** els agents s'encarreguen d'utilitzar la informació que tenen disponible per interaccionar amb els IoT.
- **Execució de serveis:** l'agent és el responsable d'utilitzar els recursos disponibles per l'execució de serveis.
- **Comunicació amb altres agents:** els agents es comuniquen entre si per a col·laborar en les execucions d'un servei, per intercanviar informació o comunicar ordres.
- **Control dels agents:** és important saber en cada moment l'estat de tots els agents degut a que si un agent cau tota la informació encarregada de rebre aquest agent es perd i no es pot utilitzar (es perd la percepció de l'estat de la nostra ciutat).
- **Control de dispositius:** s'ha de realitzar un control sobre l'estat dels dispositius IoT per tal de que aquests sempre estiguin enviant informació i actuar en cas de que es trobin inoperatius.
- **Comunicació amb el cloud:** l'agent finalment és comunica amb el cloud per delegar peticions de servei o per emmagatzemar dades.

Aquestes són les funcionalitats més importants que haurien de portar a terme els agents en el model Fog to Cloud. Com que són moltes tasques per un sol node s'hauria de simplificar el nombre de tasques per individu. Per tal de simplificar les tasques que porta a terme un agent seria necessari classificar-los. De tal manera que en aquesta capa fog hi trobaríem una divisió entre els propis nodes.

Aquesta divisió pot ser **jeràrquica** (figura 8) on uns nodes estan per sobre els altres o pot no ser-ho. De tal manera que es classificaria els nodes per **grups**, on cada grup de nodes tenen la mateixa importància.

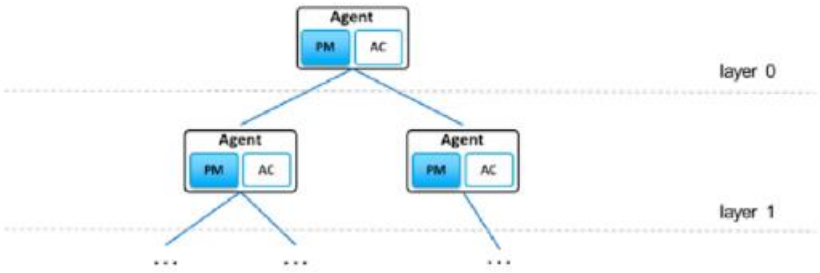


Fig. 8: Esquema de la jerarquia entre agents.

3.6 Serveis

La topologia d'una Smartcity basada en el model Fog to Cloud permet l'execució de serveis amb el recursos disponibles en temps real. Un servei és tota aplicació, eina, o funcionalitat que és efectuada correctament gràcies a la topologia de la ciutat.

A grans trets diferenciarem dos tipus de serveis: els reactius i els actius.
Els serveis **reactius** són aquells que són sol·licitats per un agent quan algun dels seus dispositius ha detectat un valor anòmal . Per exemple, una persona que compta amb un mesurador de constants vitals que detecti que els batecs/s han superat el màxim o el límit recomanat. En aquest cas es sol·licitaria un servei d'emergències.
Les peticions de servei reactives són realitzades automàticament per l'agent encarregat en qüestió.

Un altre tipus de serveis són els **actius** o aquells que sol·licita un usuari per voluntat pròpia. Per exemple, un usuari que té una aplicació per aparcar sol·licitaria una petició d'un servei d'aparcament. Aquest tipus serien els que utilitzarien els ciutadans en els seus smartphones, ordinadors...

La diferència entre aquests dos tipus de serveis és en que el primer és sol·licitat per la pròpia plataforma de la ciutat al detectar una anomalia. En canvi en el segon tipus, el servei és demanat per la voluntat d'un usuari.

4. REQUISITS

L'anàlisi dels requisits d'un projecte permet realitzar un millor disseny i desenvolupament d'aquest. En els següents apartats es detallen quins són els requisits agrupats pels **components principals** del projecte.

4.1 Requisits funcionals

Els **requisits funcionals** són aquells que a nivell de desenvolupament tècnic s'haurien d'acomplir per satisfer les necessitats del client.

4.1.1 Plataforma

Els requisits de la plataforma a desenvolupar són:

- L'ús òptim de la informació de la topologia.
- L'agregació d'agents i els seus dispositius en la topologia.
- L'execució de serveis, això involucra la gestió de peticions dels clients i el retorn de resultats.
- El funcionament de la plataforma en situacions adverses com la caiguda inesperada d'un node.

4.1.2 Agent

A continuació s'esmenten els requisits que un agent hauria d'acomplir:

- Un agent hauria d'estar dissenyat de forma modular. Cada mòdul hauria de ser independent d'un altre i tenir una sèrie de funcionalitats especifiques.
- Comptar amb mecanismes per a comunicar-se amb altres nodes de la plataforma.
- Poder registrar-se en la plataforma.
- Poder ser monitoritzat externament.
- Poder satisfer les peticions de servei del client.

4.1.3 Aplicació client

L'aplicació client d'un agent hauria de complir els següents requisits:

- Visualització dels serveis que el client pot sol·licitar.
- La realització de peticions de servei al seu agent.
- Visualització de les tasques que està portant a terme l'agent.

4.1.4 Front-end

El front-end d'administració utilitzat hauria de comptar amb les següents característiques:

- Visualització global de la plataforma, és a dir, quins nodes la formen.
- Visualització de les tasques que està realitzant cada node.
- Visualització del llistat de serveis que s'ofereixen en la plataforma.
- Visualització de la informació de cada servei.

4.2 Restriccions

4.2.1 Restricció de solució

Descripció

El front-end no serà utilitzable en mòbils o tauletes.

Justificació

Per acomplir els objectius plantejats el front-end implementat estarà dissenyat per ser visualitzat en una màquina d'escriptori com un ordinador portàtil o de sobre taula. La visualització d'aquest en màquines on la resolució és menor a la pantalla de qualsevol ordinador no serà realitzada correctament.

Descripció

El dispositiu o Device dissenyat i construït no complirà els requeriments de disseny i construcció esperats per satisfer a un client.

Justificació

Aquest projecte s'ha realitzat per un estudiant d'enginyeria informàtica sense cap coneixement en el disseny de dispositius electrònics. El disseny òptim del dispositiu no és una de les finalitats del projecte.

4.2.2 Restricció econòmica

Descripció

El projecte serà desenvolupat sense cap suport econòmic.

Justificació

Aquest projecte solament té finalitats acadèmiques.

5. DISSENY

En aquest apartat es descriu quin és el disseny obtingut de cada component pel conjunt de requisits identificats en l'apartat anterior.

5.1 Plataforma basada en Fog-to-Cloud

La plataforma dissenyada a partir d'aquest model està dividida per tres capes diferenciades: la capa Cloud , la capa Fog i finalment la capa de dispositius o d'IoT.
Tant en el nivell Cloud i Fog trobem els **agents** o nodes que participen en l'administració d'una Smartcity. De cadascun d'aquests nodes es realitza una descripció del seu funcionament i comportament en execució.

5.1.1 Cloud

El model Fog-to-cloud permet que part de les tasques que portava a terme la capa Cloud passin a ser executades per la capa Fog. Aquest canvi estructural ocasiona la reducció del nombre de màquines que participen en aquesta capa. Degut a que aquestes passaran a tenir menys responsabilitats.

En aquest projecte s'ha dissenyat una topologia on la capa Cloud compta amb una sola màquina. On trobarem un agent fixe (localitzable per altres agents) que efectuarà les tasques d'aquesta capa, l'**agent cloud**.

La principal responsabilitat d'aquest node es ajudar en la resolució d'un servei quan els agents de la capa fog no l'han pogut resoldre ja que aquest node compte amb una visió global de la topologia.

5.1.2 Fog

Com s'ha comentat anteriorment en la capa fog hi trobarem quasi tots els agents de la topologia, a excepció de l'agent cloud. Aquests agents passaran a realitzar part de les tasques que típicament realitza el cloud en el model tradicional però en la pròpia ciutat.

Els agents en el model Fog-To-Cloud són la columna vertebral de la gestió d'una Smartcity, doncs aquests nodes són els encarregats de la presa de decisions segons els recursos d'una Smartcity. En aquest projecte els agents s'han classificat segons les tasques que realitzen: distingint entre els agents "normals" i els agents leader.

S'ha establert una **estructura jeràrquica** (figura 10) basada en agents que estan uns per sobre dels altres segons el **rol** d'aquests. En el nivell superior trobarem els leaders, aquells nodes que prenen les decisions de la plataforma i en el nivell inferior es troben els agents, aquells que les porten a terme. Al nivell inferior trobem els dispositius o IoT.

5.1.2.1 Agents Leader

Els agents leader de la nostra topologia es troben en la ciutat. Una de les funcions principals d'aquest tipus agent és **enregistrar els agents** pròxims a ell que es troben en la mateixa xarxa. És a dir, que aquests nodes són els responsables d'agregar nous agents en la topologia.

Degut a que una ciutat pot tenir una grandària molt extensa aquesta s'hauria de dividir per **zones**. En cada zona es comptaria amb una subxarxa on un agent leader s'encarregaria d'agregar els agents propers a ell (fig. 9). El registre d'un agent a la topologia es produeix quan aquest es considera apte per a participar.

Com que els leaders compten amb la informació dels agents aquests són els encarregats de determinar quins nodes participen per resoldre un servei.

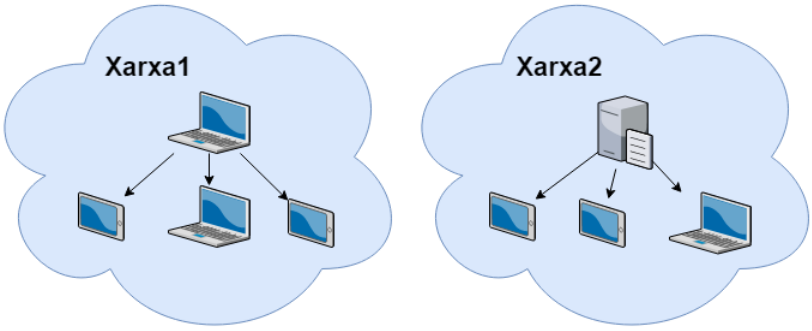


Fig. 9. Enregistrament d'agents pels leaders de la ciutat segons la xarxa on es troben.

5.1.1.2 Agents

L'agent és el component més comú de la nostra estructura i al igual forma que el leader es troba dins de la ciutat.

Els agents són agregats quan reben la petició de registre d'un leader com abans s'ha comentat. Cada agent solament es pot agregar una sola vegada a la plataforma, encara i que en una zona trobéssim dos leaders.

La classificació d'agents realitzada en la capa fog ocasiona que la plataforma presenti l'estructura que es veu a continuació:

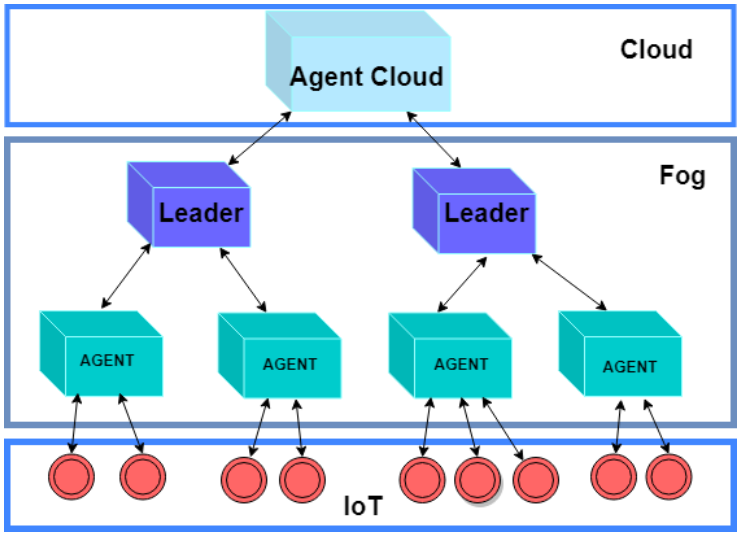


Fig. 10: Plataforma dissenyada basada en el model Fog to Cloud.

Com que aquests nodes són els més propers als dispositius aquests són els encarregats d'**executar els serveis** ja que la majoria de serveis requereixen de l'actuació d'un dispositiu.

5.1.3 IoT

La capa dels dispositius IoT és la que es situa en el nivell inferior de la jerarquia. Aquest nivell està format per dispositius dispersos per la ciutat o que es troben junt amb un agent (pot ser que un agent compti amb diversos IoT).

En aquest projecte **la capa de dispositius ha sigut simulada** en els propis agents, per tant trobarem que cada agent compta amb els mateixos tipus de dispositius.

Aquesta capa és la que en general genera el major volum de dades conjuntament amb la capa fog.
Una plataforma ha d'estar dissenyada per saber com tractar la informació dels múltiples dispositius de la ciutat.

5.1.4 Anàlisi del Pla de dades

L'estructura de nodes esmentada ens determina la distribució d'informació d'una Smartcity.

La correcta gestió de les dades d'una Smartcity és essencial per garantir als clients l'eficiència i rapidesa en les operacions de la plataforma. Degut a la importància d'aquest punt s'ha vist necessari realitzar un **anàlisi** de plans de dades diferents. Per determinar quin és el més adient segons les tasques a realitzar per la plataforma.

S'han considerat a desenvolupar fins a tres plans de dades diferents en aquest projecte: el model centralitzat, el model descentralitzat i el model topològic i local. Cadascun d'ells és descrit amb les seves principals avantatges i inconvenients. En cadascun dels següents plans de dades es detallen on s'emmagatzema la informació topològica de la ciutat.

Tots els models presentats comparteixen l'ús d'una sola base de dades on s'emmagatzema la informació dels serveis. A aquesta base de dades també es fa referència com a **catàleg de serveis**.

5.1.4.1 Model Centralitzat

El model centralitzat és aquell que com indica el seu nom es basa en desar la informació en un únic punt de la topologia. Tota la informació d'agents i dispositius presents en la topologia es trobaria en un únic punt.

Aquest pla de dades implica que tots els nodes de la plataforma haurien de realitzar connexions amb aquesta base de dades per a qualsevol operació.

Per assegurar el funcionament d'aquest model el més viable seria realitzar backups de la informació global en altres punts. Aquest model si que requeriria de més màquines en la capa cloud per tal de mantenir aquesta informació.
A continuació es veu l'estructura del pla centralitzat de dades:

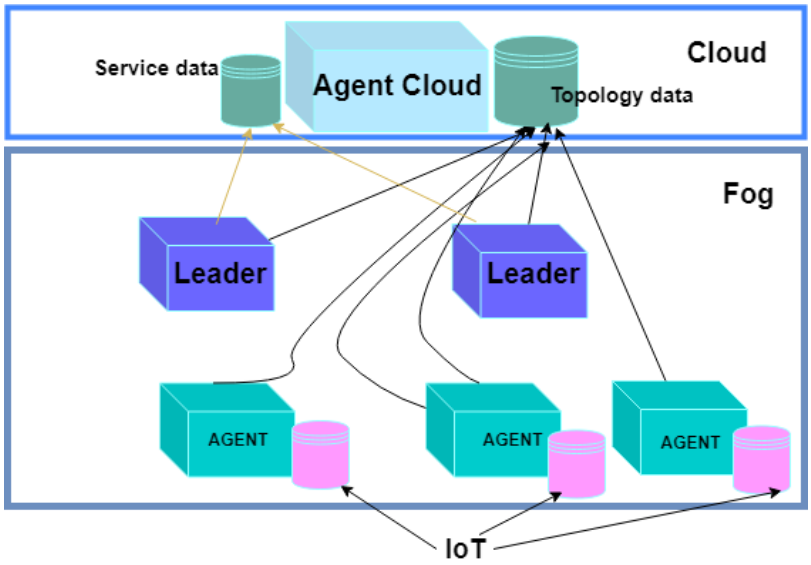


Fig. 11: Model centralitzat de dades.

Punts forts

Aquest model és **fàcilment desenvolupable** en una Smartcity. La lògica estructural és simple i radicaria el funcionament d'aquest model en l'actualització d'informació als backups. Periòdicament l'agent cloud informaria a cadascun dels backups amb part de la informació topològica.

Com a conseqüència es simplificaria també el disseny dels propis agents doncs aquests solament es connectarien a la base de dades per accedir o modificar la informació desitjada.

Inconvenients

El principal inconvenient d'aquest model és la **sobresaturació de la base de dades** si es compta amb un nombre mitjanament alt d'agents o d'IoT.
Qualsevol cerca d'informació o qualsevol operació tardaria uns quants segons en realitzar-se. Aquest retard en el temps és excessiu per una Smartcity que vol oferir eficiència i rapidesa en les execucions dels seus clients.

En cas de que es produís una caiguda de l'agent cloud la recuperació d'informació seria molt lenta. Encara que es recuperés la informació dels backups aquesta tampoc estaria actualitzada al moment actual. Doncs si han passat uns minuts, s'assumeix que la topologia de la ciutat ha canviat.

5.1.4.2 Model Descentralitzat

Aquest model es basa en dur a terme una descentralització de les dades, és a dir, en desar la informació en diferents punts de la plataforma.

Aquest model es basa en que en cada nivell desa aquella informació topològica dels nodes que estan per sota d'ell. De tal manera que l'agent cloud compta amb la informació dels leaders, els leaders la dels agents i els agents la dels IoT. A continuació és mostra l'estructura d'aquest pla de dades:

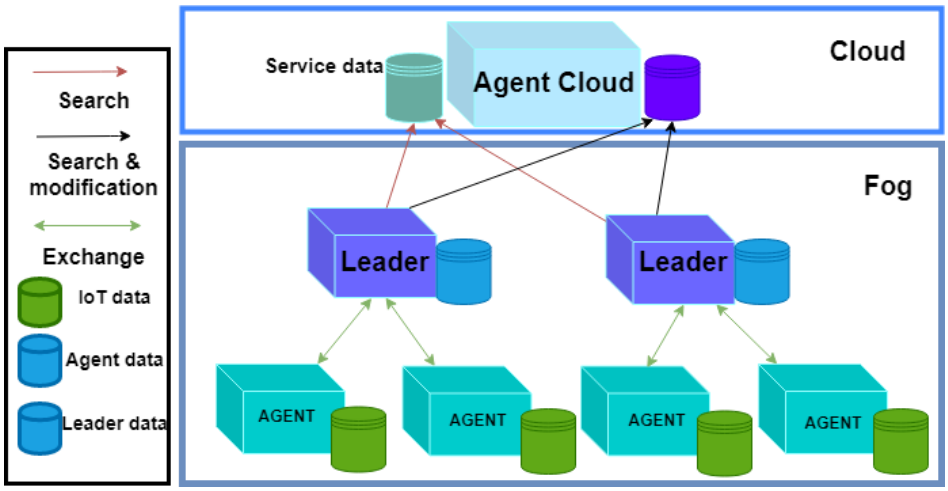


Fig. 12: Model descentralitzat de dades.

Punts forts

Pel simple fet del repartiment d'informació aquest model és més **eficient i funcional** que el model centralitzat. En aquest model no recau el pes de manteniment de les dades en un sol punt de la topologia.

El volum total d'informació està **correctament repartit**. En una ciutat amb milers de nodes i d'IoT cada node tindria la informació dels que es troben en el nivell inferior. Encara que podria succeir que hi hagués un desequilibri molt gran en la

càrrega d'informació. Per exemple que un agent comptés amb milers d'IoT i un altre amb unes desenes. O que un leader tingués en una zona molts agents i en un altre zona propera hi hagués uns pocs.

En aquest model l'agent cloud és el node que menys informació topològica guarda i per tant la màquina on s'executés no hauria de ser tant potent a nivell computacional.

Inconvenients

L'inconvenient principal d'aquest model és que la repartició de la informació està basada simplement en la jerarquia topològica de la plataforma i en evitar la repetició d'informació. Aquests fets no són inconvenients ja que són els objectius principals d'una Smartcity però en el model d'Smartcity plantejat aquest model no funcionaria.

Una Smartcity està pensada per executar serveis això implica que l'agent cloud i els leaders han de saber quins IoT tenen els agents per saber si poden executar un servei. Si aquest model fos implementat per a executar un servei es realitzarien peticions a cegues fins trobar o no agents que el poguessin efectuar.
Encara i que el volum de dades de cada node fos reduït els serveis tardarien molt en solucionar-se degut a que els leaders estarien constantment realitzant peticions als seus agents per saber si poden resoldre un servei.

Addicionalment, la monitorització d'aquesta plataforma no pot ser desenvolupada fàcilment, al no comptar amb la informació en un punt.

5.1.4.3 Model Topològic i Local

Aquest model es basa en **aprofitar el disseny de la plataforma** construïda perquè cada agent compti amb la informació que necessita i utilitza.

En aquest model l'agent cloud té la informació bàsica de cada agent de la plataforma. De tal manera que sap quins agents té un leader i quin tipus de dispositius té un agent. Els leaders també compten amb la informació dels agents i una informació bàsica dels IoT.

A continuació es veu l'estructura d'aquest pla de dades:

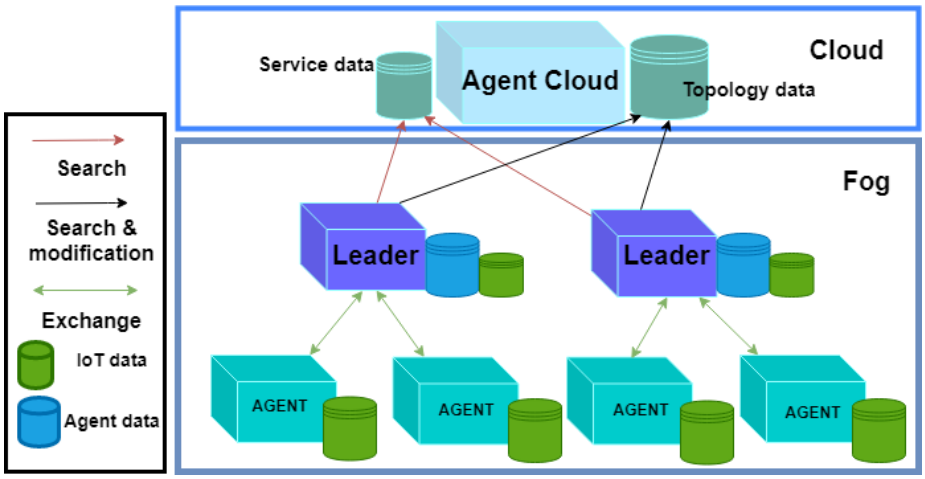


Fig. 13: Model topològic i local de dades.

Punts forts

El principal punt fort d'aquest model és que permet la resolució i execució de serveis més ràpida i efectiva respecte els model anteriors. Com que els leaders compten amb la informació dels agents i els seus IoT aquests poden resoldre els serveis sense la intervenció de l'agent cloud.

Aquest model permet que la **monitorització** de la plataforma sigui fàcilment efectuada. Solament es requereix una connexió remota per saber l'estat de la topologia.

Inconvenients

Per contra de millorar l'execució de serveis es perd unicitat de les dades. El funcionament d'aquest model implica que es produeixi la repetició d'informació. A més, aquest model és el que més **complexitat lògica i de recursos** requereix.

D'igual manera que en el model descentralitzat podria ocorre que hi hagués un **desequilibri d'informació**. Per solucionar-ho s'hauria de comptar amb un sistema que comptabilitzés la balança de càrrega per zones. En cas de que en una zona hi hagués molts agents i en un altra pocs, part de la multitud hauria de ser notificada per canviar de leader.

5.1.5 Comunicació entre els nodes

Com s'ha pogut apreciar en la figura 13 entre agents i leaders es produeixen intercanvis d'informació. Aquests intercanvis es realitzen mitjançant la comunicació entre nodes establerta en la plataforma.

La plataforma d'una ciutat intel·ligent està formada per un conjunt de nodes que es comuniquen entre si per tal de resoldre les diverses tasques d'aquesta. D'aquí esdevé que una comunicació sòlida entre nodes sigui un apartat fonamental per tal d'aconseguir una plataforma eficaç en l'administració d'una Smartcity.

En la plataforma dissenyada cada capa compta amb una **comunicació bidireccional** (fig. 10) amb el seu nivell inferior. Cada node de la plataforma es pot comunicar mitjançant un canal fiable de comunicació pel qual es tingui la confiança de que el missatge o la informació a entregar serà lliurada al receptor. De tal manera que:

- Els leaders són els únics agents de la capa fog que es poden comunicar amb l'agent cloud.
- Els agents són els nodes que es comuniquen amb els leaders i els IoT.
- Els IoT solament es comuniquen amb el seu agent. Com que en aquest projecte els IoT s'han simulat no s'ha implementat aquesta comunicació.

Aquest tipus de comunicació ens permet la transmissió d'informació entre nodes coneguts. Encara que també s'haurà de comptar amb les següents comunicacions::

- La comunicació que s'utilitza per **enregistrar un agent** a la topologia.
- La comunicació que s'utilitza per lliurar informació no gaire rellevant.
- La comunicació entre agents. És necessari incloure una **comunicació entre nodes** per a resoldre alguns serveis més ràpidament.

Addicionalment a les comunicacions definides en aquest punt s'ha incorporat una altre eina que permet la comunicació entre els mòduls del propi agent, l'API. Aquesta és definida en l'apartat 5.2.4, en el disseny d'un agent.

5.1.6 Tolerància a caigudes

Com es pot veure en els dos apartats anteriors els leaders són els que actuen com a pilar del funcionament de la plataforma. Són els encarregats d'agregar nous agents i dispositius i permeten comunicar/delegar les peticions dels agents.
A més, a nivell comunicatiu actuen com a pont comunicatiu entre els nivells de sota (agents i IoT) i el cloud.

Es pot deduir doncs que el problema principal del disseny de plataforma presentat fins al moment és que hi ha una **forta dependència dels leaders** pel correcte funcionament de l'estructura.

Quan un leader surt de la ciutat voluntàriament notifica als seus agents i aquests passen a enregistrar-se amb un altre leader.
El pitjor cas que es pot donar és que un leader abandoni la plataforma però els agents no siguin capaços de detectar-ho. En aquest cas els agents continuarien realitzant peticions a un leader que ja no es troba operatiu i el cloud donaria ordres al leader que no es portarien a terme.
Aquest abandonament involuntari pot ocórrer per múltiples causes: una tallada de llum, el reinici de la màquina , l'esgotament d'una bateria...

Aquest problema ha derivat en el desenvolupament d'un **sistema anti-caigudes** que detecti i actuï enfront la caiguda de qualsevol agent de la ciutat. A excepció de l'agent cloud, en aquest projecte s'ha assumit que aquest node no pot abandonar la plataforma.

5.1.6.1 Detecció de caigudes

Els agents leaders de la nostra ciutat són els responsables de la detecció de caigudes dels seus agents. Això permet que la topologia de la nostra ciutat sigui coneguda en tot moment per saber quants agents i dispositius es compten.

Per poder assegurar que tenim un sistema robust caldrà assegurar que es realitza una detecció de la **caiguda del agent leader**.

Per a procedir davant la caiguda d'un leader s'escull a un dels nodes de la zona com el responsable d'actuar davant la caiguda d'un leader, l'agent backup.

5.1.6.2 Agent backup

Per tal d'assegurar el bon funcionament de la Smartcity s'ha assignat un nou rol entre els agents: el rol de **backup** o de suport. Un dels molts agents que té assignats un leader serà el responsable de detectar i actuar en conseqüència quan es detecti la caiguda del leader. Encara i tenir aquesta responsabilitat addicional aquest node té les mateixes responsabilitats d'un agent qualsevol.

Aquest agent backup serà un agent qualsevol que el leader escull en algun moment donat, quan aquest agent forma part de la plataforma. Si en algun punt es detecta la caiguda de l'agent backup el leader escolliria un de nou.

L'aparició d'aquest nou rol no afecta en la jerarquia prèvia establerta com es pot veure en la següent imatge:

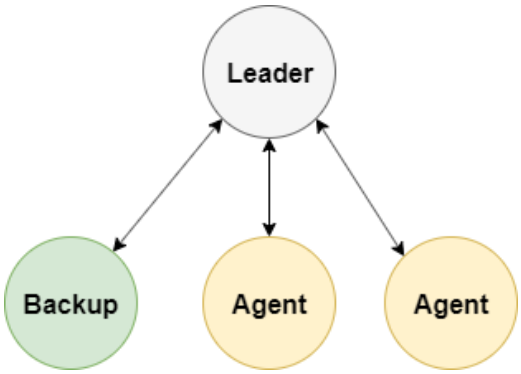


Fig. 14: Esquema dels components de la capa fog.

5.1.6.3 Caiguda del leader

Cada agent s'encarrega de detectar el funcionament del leader sigui backup o no. Quan un leader es considera que no es troba operatiu l'agent **backup passarà a ser el nou leader** de la zona.

El procediment que es segueix davant la caiguda d'un leader és el següent:

1. Un agent s'enregistra amb el leader més pròxim.
2. El leader li comunica que és l'agent backup.
3. El leader per algun motiu i sense previ avís cau.
4. Els agents de la zona ho detecten i es posen a l'escolta de peticions de registre.
5. L'agent backup passa a ser el nou leader de la zona actualitzant la informació de la base de dades topològica.
6. Els agents que es trobaven assignats al primer leader es registren de nou amb el nou leader.

Aquests procediment s'efectuaria cada vegada que es produís la caiguda d'un leader per l'agent backup assignat. Aquest **procediment assegura el funcionament de la plataforma** davant la caiguda de qualsevol leader de la ciutat.

5.1.7 Monitorització de la plataforma

Les tasques que realitza cada agent es mostren per la pantalla de l'aplicació client per informar a l'usuari. Aquesta informació també s'hauria de poder visualitzar d'alguna forma en el front-end de l'administrador.

Qualsevol disseny de plataforma d'una ciutat intel·ligent ha de comptar amb un sistema que permeti monitoritzar els nodes de la plataforma. A continuació es detalla com s'ha aconseguit dur a terme una monitorització dels agents que formen part de la plataforma d'una ciutat intel·ligent.

5.1.7.1 Administrador

Per tal de monitoritzar la topologia en un moment donat s'ha definit una **màquina** que pot trobar-se fora ciutat. Aquesta màquina la anomenarem "administrador" i serà on es trobi el **front-end** de la plataforma.

5.1.7.2 Enviament de dades

Per tal de monitoritzar a cadascun dels agents de la nostra ciutat aquests han d'informar a l'administrador. Per a poder ser monitoritzat cada agent hauria de d'enviar la seva informació amb les tasques que està portant a terme.

Aquesta informació s'hauria d'enviar periòdicament a l'administrador. Des del front-end s'hauria de poder visualitzar aquesta informació en temps real conforme un agent va realitzant tasques.

5.2 Agent

Prèviament al disseny i la implementació d'un agent s'ha realitzat una investigació de llibreries i protocols per a establir les comunicacions de la plataforma.
L'explicació de cada llibreria i la justificació d'us de la llibrera utilitzada finalment per a les comunicacions es pot trobar en un annex del projecte (apartat 15.1 Estudi de llibreries en python3).

En aquest apartat es descriuen els **mòduls principals** d'un agent.

5.2.1 Disseny modular

Per tal d'implementar un agent el més adaptable possible cal realitzar un disseny modular el més correcte possible.
Cal esmentar que el següent model presentat és una possible solució, però no té perquè ser la única ja que segons les necessitats del disseny es poden modificar o afegir mòduls.
El diagrama modular és el següent:

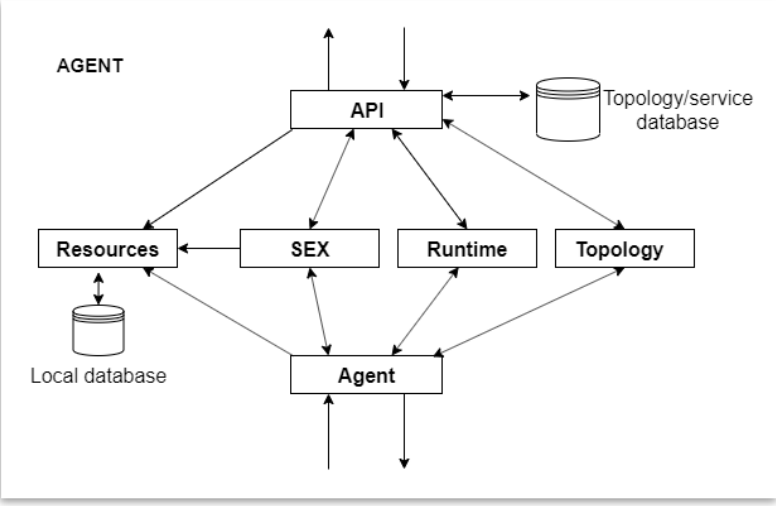


Fig. 15: Diagrama modular d'un agent.

5.2.2 Aplicació client

L'aplicació client o el mòdul **app_agent** és des del qual es posa en marxa un agent. En aquest mòdul es generen i es passen els paràmetres d'execució inicial d'un agent:

- **Direcció IP:** l'adreça IP del node.
- **Port:** El port base d'un agent per a l'establiment de comunicacions. Aquest està indicat per defecte i ha de ser diferent del port que utilitza l'API. Aquest paràmetre al igual que la direcció IP pot ser indicat per a la realització de proves.
- **Rol**: un agent pot actuar com cloud, leader o agent qualsevol. Aquest paràmetre és l'únic obligatori a indicar.

Aquesta aplicació no requereix de complexitat en quant a l'estètica. Solament mostra a l'usuari un llistat de serveis que pot sol·licitar. L'usuari solament hauria de marcar o seleccionar el servei a sol·licitar amb la introducció de dades corresponent (si el servei en qüestió ho requereix).

A nivell intern les peticions de servei que realitza el client es realitzen a l'agent que s'està executant. Aquestes peticions es realitzen a través de l'API al mòdul service execution del agent en qüestió. Aquests dos mòduls són descrits posteriorment.

En la següent imatge es representa com s'efectua la petició de servei del client al agent:

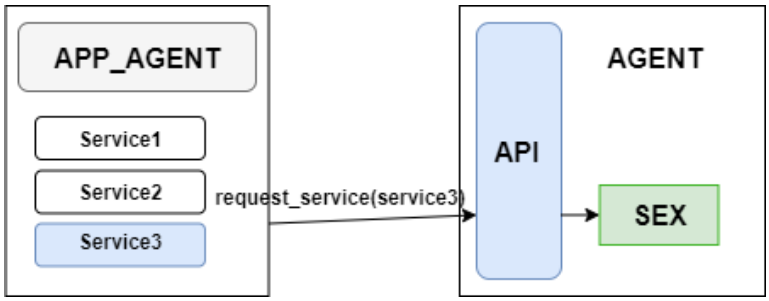


Fig 16: Esquema conceptual de l'aplicació client i el seu agent.

5.2.3 Agent

Aquest és el mòdul principal del que entenem com a un agent. Com s'ha comentat anteriorment un agent està format per un **conjunt de mòduls** on aquest és un més (no confondre pel nom).

En la següent taula es descriuen els atributs principals d'aquesta classe:

Atribut	Tipus	Descripció
ID	String	Identificador únic del node..
IP	String	Direcció IP amb la que s'executa un agent.
Port	Int	Port d'escolta base.
Role	String	Rol d'un agent, aquest pot ser: cloud, leader i agent o backup.
Connections	Dictionary	Diccionari que com a clau utilitza els identificadors dels agents i com a valors els sockets de comunicacióTCP.
Connected	Bool	Indica l'estat de connexió d'un agent normal. Es troba a True si un agent es troba enregistrat amb un leader i false si aquest agent està a l'escolta de peticions.
Backup	Bool	Booleà que indicà si un agent és backup o no.
IP leader	String	Direcció IP del leader pel qual un agent està connectat. Camp buit en els leader i el cloud.
Order	Dict	En un servei complexe els agents emmagatzemen el servei a resoldre (la seva informació) per a realitzar l'execució quan l'agent coordinador dóna l'ordre

Taula 3: Atributs principals de la classe agent.

En aquest mòdul hi consten totes les funcions d'establiment i d'acceptació de connexions, d'escolta de missatges o d'informació i de comprovació d'estat de les connexions entre agents.

Des d'aquest mòdul es posen en marxa els mòduls que han d'estar en constant execució com l'API. Conjuntament amb l'API aquest mòdul permet la **comunicació amb altres nodes**.

5.2.4 API

L'API és un mòdul que s'encarrega de rebre peticions HTTP. Aquest mòdul permet portar a terme una **comunicació entre nodes** sense requerir d'un canal de comunicació.

En la següent taula es defineixen breument els atributs principals d'aquest mòdul:

Atribut	Tipus	Justificació
Agent	Agent	El propi agent és requerit per realitzar determinades accions des de l'API.
Topology	Client Mongodb	Aquest atribut s'utilitza per realitzar operacions sobre la base de dades topològica.
Services	Client MongoDB	Aquest atribut s'utilitza per realitzar operacions sobre el catàleg de serveis.

Taula 4: Atributs principals de la classe API.

Aquest mòdul com es pot veure en la figura 4 és el que permet l'accés a les bases de dades.

Com a variables locals a destacar es defineixen el **host** o l'adreça ip del host de la base de dades topològica i de serveis. També el **port d'escolta** del servei MongoDB de l'agent cloud. Com a variable global de l'aplicació s'utilitza el port de l'API, que per defecte és el mateix en tots els nodes.

L'API definida en aquest projecte compta amb els següents tipus de peticions:

- **Delete:** petició utilitzada per eliminar informació.
- **Get:** petició utilitzada per obtenir informació.
- **Post:** petició usada per afegir informació.
- **Put:** petició usada per actualitzar la informació d'un node.

Aquesta API s'ha incorporat per actuar com a **eina intermediària entre els mòduls** d'un agent. De tal manera que permet la comunicació entre mòduls de dos nodes.

En la següent imatge es mostra com es produeix la comunicació entre dos mòduls:

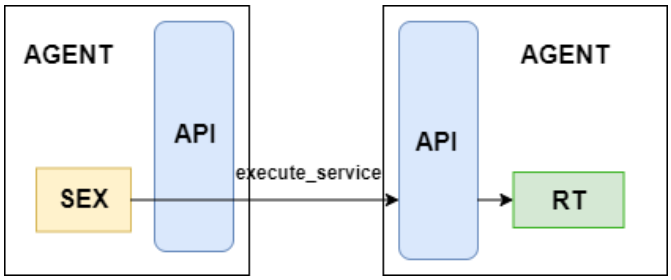


Fig. 17: Comunicació entre els mòduls Service Execution i Runtime a través de l'API.

5.2.5 Action

Aquest és un mòdul auxiliar del mòdul Agent. Hi consten els mètodes que s'utilitzen més d'una vegada en la classe Agent i mètodes més secundaris de la classe agent: com la validació d'identificadors o l'obtenció de paràmetres de sistema.

5.2.6 Topology

Aquest és el mòdul que permet realitzar operacions de modificació o consulta a la base de dades topològica o de serveis a través de l'API.

5.2.7 Resources

Aquest és el mòdul encarregat d'interaccionar amb la **base de dades local** d'un agent. Solament consten els mètodes de consulta i de modificació sobre la base de dades local. Aquest mòdul solament pot ser accedit des de:

- Mòdul agent i action: per agregar, eliminar o actualitzar la informació d'agents o IoT.
- Service execution: per actualitzar i obtenir la informació d'agents que poden resoldre un servei.
- API: per actualitzar la disponibilitat d'un agent.

5.2.8 Service Execution (SEX)

Aquest mòdul s'encarrega de la gestió de peticions de serveis i per tant es porta a terme la seva resolució. Les peticions poden ser rebudes des de l'aplicació client o d'un altre agent.

5.2.9 Runtime

Cal diferenciar-aquest mòdul del service execution comentat anteriorment. Doncs en el primer es realitza la senyalització que requereix la execució d'un servei i en aquest es realitza la pròpia execució d'aquest.
Aquest mòdul es l'encarregat d'**executar un servei** realitzant la descàrrega dels fitxers oportuns.

5.2.10 Mòduls addicionals

Tots els tipus de **dispositius** simulats compten amb una classe, on es defineixen els atributs del mòdul en si. Per exemple, d'un semàfor hi consten l'identificador i el color. S'han utilitzat un mòdul pels semàfors, per les barreres i per les ambulàncies.

5.3 Serveis

En aquest apartat es detalla que és un servei, quins tipus hi ha, com es resolen i com finalitzen.

Un servei és el conjunt d'**accions** que es duen a terme en una Smartcity per acomplir un objectiu determinat. En aquest projecte es veuen els serveis com l'execució d'un conjunt de fitxers **executables** que porten a terme unes accions determinades. Aquestes accions poden ser des de canviar l'estat d'un dispositiu fins al càlcul o computació d'un resultat.

Com s'ha comentat anteriorment la definició d'un servei es guarda en el **catàleg de serveis** de l'agent cloud. En aquesta base de dades es guarda la definició de cada servei. En aquest projecte s'ha utilitzat la següent definició:

Camp	Tipus	Descripció
Identificador de servei	String	Identificador únic de servei.
Descripció	String	Descripció breu del servei.
Codi	String	Codi principal del servei a executar.
Dependències	List	Llista amb els identificadors de serveis que depenen dels resultats del servei principal.

IoT	List	Llista amb els tipus de dispositius requerits per executar el servei.
Recursos (Resources)	List	Llista dels recursos principals que necessita un servei per ser executat correctament (RAM,CPU...).
Paràmetres	List	Llista de valors que s'utilitzen en l'execució del codi principal. Aquests poden ser generats en la petició client o definits per defecte.
Extensió programa (Program_version)	String	Indica l'extensió del codi a executar: java, python3, python2...
Codis dependents	List	Llista amb el nom de cadascun dels fitxers que necessita el codi principal prèviament a l'execució.

Taula 5: Definició d'un servei.

La definició esmentada és la fitxa bàsica de com s'hauria de definir un servei per poder ser executat en la plataforma dissenyada. Si un servei és més complexe i requereix d'una definició més àmplia solament s'haurien d'afegir els **camps addicionals.** Qualsevol servei pot ser executat sempre i quan segueixi la definició bàsica indicada.

El disseny de la plataforma i dels serveis permet que qualsevol servei pugui ser executat en la plataforma si el servei es classifica de les següents formes esmentades.

5.3.1 Tipus de serveis

En aquest projecte s'han classificat els diversos serveis segons el mètode de resolució de cadascun. A continuació es defineixen els tres tipus de serveis contemplats: els serveis simples, els serveis dirigits i els serveis complexes.

5.3.1.1 Serveis simples

Identificarem com a serveis simples aquells que poden ser efectuats per un o diversos agents. L'execució d'aquests serveis s'efectua de forma **independent**, és a

dir, un agent duu a terme l'execució sense necessitat de comunicar-se amb altres agents per acomplir l'acció encomanada.
Si un servei simple és resolt per diversos agents voldrà dir que aquests comparteixen alguna característica comú: per exemple que disposin del mateix tipus d'IoT al càrrec.

5.3.1.2 Serveis dirigits

Aquests tipus de serveis són aquells que es resolen amb algun tipus d'indicació de qui pot resoldre el servei. En la definició pròpia del servei s'inclou algun camp que indica informació dels agents que el poden resoldre. Aquesta informació pot ser l'adreça IP d'un agent sempre present a la plataforma o el seu identificador únic.

En un entorn d'Smartcity aquests serveis serien els **serveis reactius** que es duen a terme per agents coneguts de la ciutat. El node encarregat de la resolució s'estalvia realitzar una cerca si els agents indicats poden resoldre el servei.

La principal diferència que tenen respecte aquests respecte els simples és que aquests es resolen més ràpidament. Es consideren una variant dels serveis simples.

5.3.1.3 Serveis complexes

Els serveis complexes són aquells que es resolen mitjançant la **comunicació i coordinació** entre diversos agents. Aquests tipus de serveis es resolen entre diversos agents degut a que el resultat de l'execució principal és requerit per executar els serveis posteriors.

El procediment de resolució d'aquests serveis és el següent:

- Primerament s'escull un **agent coordinador** de l'execució que s'encarrega d'executar el servei principal.
- Es resol cada servei dependent del principal de la mateixa forma que un servei simple.
- Els agents estableixen les comunicacions amb l'agent coordinador per posteriorment executar el servei assignat.

L'execució d'aquest servei involucra que s'efectuï una **comunicació entre agents**.

5.3.2 Peticions de servei

Una petició de servei és la **demanda de la resolució** d'un servei sol·licitada per un agent de la ciutat (un client). En una Smartcity real la gran majoria de peticions de servei poden ser realitzades voluntàriament o com a conseqüència d'un canvi en els IoT dispersos per la ciutat.

Les peticions de servei poden ser realitzades pels agents i leaders a través de l'aplicació client que porta incorporat el software d'un agent.

5.3.3 Resolució d'un servei

La petició de servei enviada d'un agent solament pot ser resolta pels leaders de la ciutat i l'agent cloud. Per tal de resoldre un servei els leaders i l'agent cloud utilitzen uns camps addicionals que s'inclouen en la informació del servei per facilitar la seva resolució. Aquests camps són els de **senyalització del servei** i són els que es poden veure en la següent taula:

Camp	Tipus	Descripció
IP Agent Request	String	L'adreça ip del client sol·licitant del servei, s'adjunta per a la notificació de resultats.
Location	List	Llista que conté les credencials necessàries per obtenir els codis dels serveis del servidor cloud. El host, l'usuari, contrasenya i directori remot comprenen aquesta llista.
IP target	String	Adreça ip de l'agent coordinador.
Port	Int	Port base d'escolta de l'agent coordinador.
Output	List	Llista amb les adreces ip o identificadors dels agents que participen en un servei.

Taula 6: Camps de senyalització d'un servei.

Els camps IP target i Port solament s'utilitzen en la senyalització d'un servei complexe.

5.3.3.1 Leader

Quan una petició de servei és rebuda s'analitza si aquesta petició pot ser resolta o no. El leader és el responsable de determinar primer qui pot resoldre el servei sol·licitat segons els requeriments d'un servei.

Els passos per a resoldre un servei són els següents:

1. Determinar quin tipus de servei es tracta: simple, dirigit o complexe.
2. Determinar quins són els agents que poden resoldre el servei.
3. Notificar als agents corresponents aportant la informació del servei necessària.

Aquesta informació esmentada està formada per la informació que aporta el client al sol·licitar el servei i per la informació del servei que es troba en el catàleg de serveis.

5.3.3.2 Cloud

L'agent cloud és l'encarregat de resoldre un servei quan un leader de la ciutat no compta amb agents disponibles per executar el servei demanat. Aquest agent a diferència dels leaders duu a terme una cerca molt simple dels nodes capaços de resoldre un servei. S'ha de tenir en compte que la base de dades topològica és significativament més gran que la local d'un leader.

Quan l'agent cloud troba un **agent d'una altra zona** que pot resoldre el servei li delega al seu leader la resolució. En la zona del leader al que se l'hi delega la resolució es tindrà la certesa que almenys un dels nodes és capaç d'executar el servei.

5.3.4 Execució d'un servei

L'execució d'un servei es produeix quan l'ordre és donada directament pel leader en els serveis simples o bé per un altre agent en els complexes.
Sempre s'assumeix que l'agent que executa un servei no compta amb cap fitxer del servei.
Per executar un servei l'agent utilitza la informació proporcionada pel leader per a descarregar tots els fitxers necessaris.

Un cop l'execució finalitza l'agent s'encarrega de notificar amb els resultats al leader i al client.

5.3.5 Finalització d'un servei

Un servei es dóna per finalitzat quan totes les execucions han finalitzat, les comunicacions establertes s'han tancat i els resultats han sigut notificats al client.

La notificació de resultats sempre es portada a terme per un sol agent. Sempre hi haurà un responsable en la plataforma de notificar el resultat al client però no en pot haver diversos si el servei s'ha efectuat correctament.

5.4 Disseny del front-end

En aquest apartat es detalla com s'ha dissenyat el front-end utilitzat en la monitorització de la plataforma.

El front-end dissenyat hauria d'estar dividit en dos apartats principals: l'apartat topològic i el de serveis. Aquest front-end hauria de comptar 4 pantalles diferents:

- Pantalla de la **visualització topològica**: aquesta permet a l'usuari saber a primera vista quins nodes formen la plataforma en el moment actual.
- Pantalla d'**informació d'un node**: aquesta permet la visualització de la informació d'un agent. Aquesta pantalla és diferent per leaders i agents.
- Pantalla de la **visualització dels serveis**: aquesta pantalla permet la visualització del llistat de serveis disponibles pels usuaris.
- Pantalla d'**informació d'un servei**: aquesta pantalla permet veure la fitxa bàsica d'un servei que utilitzen agents i leaders.

5.4.1 Pantalla principal

En la següent imatge es mostra un disseny simple del que podria ser la pantalla de visualització topològica o dels serveis:

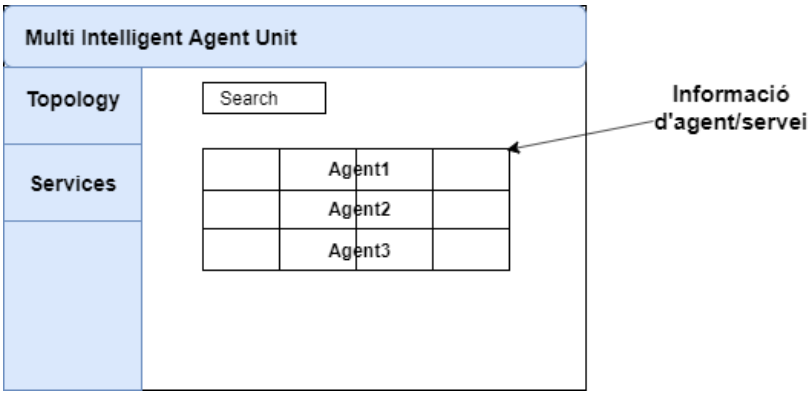


Fig. 18: Disseny de la pantalla de visualització d'agents i serveis

5.4.2 Pantalla d'informació d'un node

La **pantalla de visualització d'informació d'agents** i leaders hauria de ser diferent per cada node ja que aquests guarden diferents tipus d'informació. Encara que es poden diferenciar tres apartats clars en el disseny de la pantalla:

- Apartat dedicat a mostrar la informació del node.
- Apartat dedicat a mostrar la informació pròpia del node. En el cas dels leaders es mostraria la informació dels agents i en el dels agents la informació dels IoT.
- Apartat dedicat a mostrar la informació d'execució.

La següent imatge mostra com hauria d'estar dissenyada aquesta pantalla:

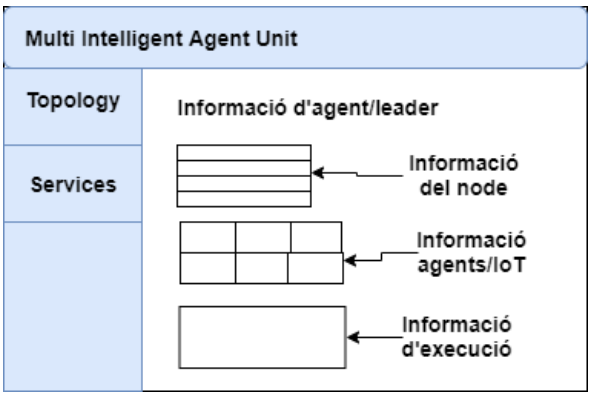


Fig. 19: Disseny de la pantalla de visualització d'informació d'un agent.

5.4.3 . Pantalla d'informació d'un servei

La pantalla de visualització de la informació de serveis podria ser com l'apartat que mostra la informació bàsica d'un agent. Aquesta pantalla no compta amb cap requisit de visualització específic.

En aquesta pantalla s'hauria de mostrar la mateixa informació que està definida en el catàleg de serveis.

6. DESENVOLUPAMENT TÈCNIC

En aquest apartat es descriu el resultat de la implementació del disseny proposat en el capítol anterior dels components principals.

Aquest apartat comença amb un resum de tecnologies i eines utilitzades per portar a terme el projecte. Posteriorment es realitza una explicació tècnica de la implementació de la plataforma, dels agents, dels serveis i del front-end.

6.1 Resum de tecnologies

6.1.1 Plataforma

Les tecnologies utilitzades de la plataforma són:

- **MySQL** com a sistema d'emmagatzematge local de dades.
- **MongoDB** com a sistema d'emmagatzematge de dades de l'agent cloud.
- **SFTP** com a servei de pujada i descàrrega dels fitxers de dades.

6.1.2 Agents

Les tecnologies utilitzades dels agents són:

- **Python3** com a llenguatge de programació.
- **Cherrypy** com a framework web de l'API.
- Llibreria **Socket** com a canal de comunicació entre agents.
- Llibreria **Threading** per la creació i gestió de processos.
- Llibreries de python:
    - Pysftp: per realitzar connexions amb el servidor SFTP.
    - Requests: per realitzar peticions HTTP a l'API d'un altre agent.
    - Pymongo: per realitzar connexions amb una base de dades MongoDB.
    - Psutil: per obtenir informació dels paràmetres de sistema.
    - Mysql-client: per realitzar connexions mb un base de dades Mysql

6.1.3 Front-end

Les tecnologies del front-end són:

- Servidor web: Apache2.
- Codi en HTML5,PHP i JavaScript.
- Framework: Bootstrap.

6.2 Eines de desenvolupament

A continuació es descriuen breument les eines de desenvolupament utilitzades en aquest projecte:

6.2.1 VM VirtualBox

Virtualbox és una aplicació que permet la creació i gestió de màquines virtuals amb diferents sistemes operatius en una sola màquina física. Aquest projecte requereix la disponibilitat de diverses màquines diferents per a la realització de proves. Amb aquesta aplicació s'han realitzat les proves de connexió i comunicació entre màquines diferents ja que VirtualBox permet l'execució de diverses màquines virtuals al mateix temps.

6.2.2 Editors de text

L'aplicació dels agents ha sigut desenvolupada amb l'editor de text Gedit. Per a la modificació de fitxers de configuració dels servidors configurats s'ha utilitzat l'editor nano. Els dos editors són propis de Linux i Debian.

6.2.3 Navegadors web

S'han utilitzat els navegadors web Mozilla Firefox i Google Chrome per a la visualització del front-end i per realitzar proves de funcionament de les APIs dels agents.

6.2.4 MySQL i MongoDB

Per comprovar el funcionament de les operacions relacionades amb les bases de dades s'han utilitzat els gestors de bases de dades de MySQL i MongoDB. Aquests han sigut utilitzats a partir del propi terminal de Linux.

6.2.5 PhPStorm

S'ha utilitzat aquest software per desenvolupar i dissenyar el front-end de l'administrador. És un editor de text que està pensat per ser utilitzat pel desenvolupament web i permet realitzar connexions des de la pròpia aplicació al servidor web de la màquina.

6.3 Plataforma

La plataforma s'ha implementat de forma que aquesta s'estableix de forma automàtica a partir de l'execució dels agents que la formaran.

Per descriure com s'estableix i funciona la plataforma en execució es descriu el **mòdul Agent** per cada tipus de node. Com anteriorment s'ha comentat en aquest mòdul hi consten tots els mètodes que estableixen les comunicacions entre nodes.

El mòdul Agent s'executa de l'aplicació client **app_agent.py**. L'execució satisfactòria d'un agent solament requereix la indicació del rol del node. La direcció ip utilitzada s'agafa automàticament de les interfícies de xarxa actives de la màquina. En cas de que aquesta obtenció de l'adreça no funcioni l'usuari és notificat ja que l'agent ha de comunicar-se amb altres nodes.

6.3.1 Passos previs a l'establiment

Cada node prèviament a l'execució realitza unes tasques que asseguren que l'establiment de la plataforma es realitzi de forma adequada. A continuació s'esmenten quines són aquestes tasques depenent el rol de cada node.

6.3.1.1 Cloud

El primer node que s'hauria d'executar és l'agent cloud. Aquest node s'encarrega de preparar la plataforma per a l'execució de serveis mitjançant l'execució d'un programa d'inicialització.

Init_services.py

Aquest programa s'encarrega de carregar la informació dels serveis de la plataforma en el catàleg de serveis (al MongoDB) cada vegada que és executat. Addicionalment, el conjunt de fitxers dels serveis són pujats al servidor SFTP del propi agent cloud.

En cas de que alguna càrrega d'informació no s'hagi realitzat correctament l'usuari és notificat amb el que ha succeït.

6.3.1.2 Leaders

En aquest projecte s'ha assumit que cada node es pot executar més d'una vegada en la mateixa màquina. Per tal de que el leader realitzi les seves tasques adequadament aquest esborra tota la informació de la seva base de dades local MySQL(tant taules com dades). Quan el leader pot emmagatzemar informació d'agents i dispositius adequadament passa a efectuar les seves tasques.

6.3.1.3 Agents

Cada agent s'inicia amb la **generació dels IoT** al seu càrrec en la seva base de dades local (10 semàfors, 10 ambulàncies i 10 barreres). D'igual forma que el leader aquest s'encarrega de esborrar tota la informació prèvia en les taules de MySQL per generar nous dispositius correctament.

Aquesta neteja d'informació i taules realitzada tant per leaders com agents es correspon al mètode **create_topology** del mòdul resources. L'agent solament compta amb les taules dels ToT encara que aquestes guarden més informació que la dels leaders (pels semàfors el color, per les ambulàncies les coordenades, etc...).

6.3.2 Establiment de la plataforma

Com s'ha esmentat anteriorment el mòdul Agent permet l'establiment de la plataforma, per això és el que es posa en marxa primer. En la següent imatge es mostra el mètode constructor de la classe Agent:

```
def __init__(self, id_agent,ip, ip_agent, port,connexion,role,backup=None):
    self.id = str(id_agent)
    self.ip = str(ip)
    self.ip_leader = str(ip_agent)
    self.port = int(port)
    self.status = True
    self.backup=backup
    self.role=role
    self.num = 0
    self.connections={}
    self.agents = []
    self.connected=False

    if connexion:
        action.create_json(self)    # agent data in json format
        if self.role == "agent":
            self.bind_agent()       # start listening to leader registration requests

        elif self.role == "leader" or "cloud":
            self.realise_binding()  #

        self.run()                  # RUN daemon threads
        self.API=api.API(self)
        self.API.start()            # RUN AGENT API
```

Fig. 20: Mètode constructor de la classe Agent

Per tal de descriure l'establiment de la plataforma es descriuen primer els mètodes que són usats per la gran majoria d'agents (amb alguna excepció de rol):

Mètode realise_binding

Aquest mètode és utilitzat pels leaders i el cloud per establir les comunicacions necessàries. Cada node al executar-se estableix les comunicacions adients mitjançant **sockets**:

- **Cloud:** l'agent cloud compta amb un socket TCP pel canal de comunicació amb els leaders.
- **Leaders**: compten amb dos sockets TCP per a la comunicació amb el cloud i agents i un socket UDP. Aquest socket UDP s'utilitza per enregistrar nous agents en la plataforma i per rebre/enviar informació que no sigui prioritària.

Cada socket s'estableix en la direcció ip del node però en ports diferents.

Mètode bind_agent

Aquest mètode solament és utilitzat pels agents "normals" i es basa en establir el socket UDP per escoltar missatges enviats a qualsevol adreça de la màquina.

Mètode run

En aquest mètode cada agent executa els seus processos threads adients en mode **daemon**, és a dir, en background. Cada procés s'executa indefinidament. En la següent imatge es mostra la creació i declaració d'aquests processos:

```
accept_leaders = threading.Thread(target=self.accept_connexions_leaders)
accept_leaders.setDaemon(True)
accept_leaders.start()
readf_leaders = threading.Thread(target=self.readf_leaders)
readf_leaders.setDaemon(True)
readf_leaders.start()
```

Fig. 21: Mòdul Agent. Execució de processos en mode daemon de l'agent cloud.

Tots els nodes que són monitoritzats compten amb el procés send_data executant-executant-se en segon pla a excepció del cloud.

Mètode send_data

Aquest mètode està implementat per a que l'agent en qüestió pugui ser monitoritzat un cop s'agregi a la plataforma.

En el moment inicial d'execució cada agent genera el seu **fitxer de text** corresponent a les tasques que està realitzant (s'escriuen els missatges generats al fitxer i es sobreescriu) i l'envia a l'administrador. En aquest fitxer de text (com a nom s'utilitza l'identificador únic del node) hi consta la mateixa informació per pantalla que veu l'usuari.

La càrrega d'aquest fitxer al servidor SFTP de l'administrador es realitza mitjançant l'execució del programa **upload.py**. Aquest procediment s'efectua encara i que la màquina de l'administrador no es trobi operativa.

A continuació es descriuen els processos/mètodes més importants de cada agent.

6.3.1.1 Agent Cloud

L'agent cloud compta amb dos processos (figura 21) executant-se indefinidament: l'acceptació de connexions dels leaders i l'escolta de missatges. Aquesta acceptació i escolta es realitzen mitjançant el socket TCP d'aquest node esmentat anteriorment. Aquest node s'encarrega d'eliminar tota la informació de la base de dades topològica per si s'ha executat anteriorment (mitjançant la crida delete_agents a la seva API).

Com que solament els agents leaders verídics que formen part de la plataforma coneixen la ubicació de l'agent cloud no es comprova l'autenticitat d'aquests.

Mètode accept_connexions_leader

Aquest és el mètode que l'agent cloud usa per establir la plataforma mitjançant la comunicació amb els leaders.

El cloud ha d'estar constantment a l'escolta de la informació que envia un leader en qualsevol moment donat. Per això quan un leader es connecta amb el cloud aquest node emmagatzema el socket TCP usat en el diccionari de connexions. Com a clau s'utilitza l'identificador del leader. Seguidament la informació del leader s'afegeix a la base de dades topològica a través del mòdul topology.

6.3.1.2 Agents leaders

El leader compta amb 3 processos relacionats amb l'establiment de la plataforma: la connexió amb el cloud, la captació d'agents i l'acceptació de connexions.

Mètode connexion_cloud

Per facilitar el desplegament de la plataforma els leaders compten amb un thread dedicat a la connexió automàtica amb l'agent cloud. Cada 5s cada leader s'intenta connectar amb l'agent cloud fins que aquest es trobi operatiu. Aquesta connexió es realitza com a client TCP (el cloud actua com a servidor) amb la adreça ip i port d'aquest node, que es troben definides com a variables.

Mètode Register_agents

Per a la captació i posteriorment enregistrament d'agents tant leader com agent han d'utilitzar el mateix port base de comunicacions.

Per captar nous agents un leader envia periòdicament **missatges a l'adreça broadcast** de la xarxa en la que es trobi. Aquests missatges s'envien mitjançant el socket UDP del leader. En la imatge següent es mostra el fragment de codi que permet configurar el socket UDP (socket_broadcast) per enviar els missatges a aquesta adreça :

```
if self.role == "leader":
    self.socket_leader =  socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socket_leader.bind((self.ip, self.port))
    self.socket_leader.listen(MAX_NUM_AGENTS)
    self.socket_broadcast =  socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.socket_broadcast.setsockopt(socket.SOL_SOCKET,socket.SO_BROADCAST,1)
    self.socket_broadcast.bind((str(self.ip), int(self.port)+1))
    self.socket_cloud =  socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socket_cloud.bind((str(self.ip), int(self.port)+2))
    self.socket_cloud.listen(MAX_NUM_AGENTS)
```

Fig. 22: Mòdul Agent. Establiment dels sockets de comunicació del leader.

Per tal de no congestionar la xarxa aquests missatges s'envien periòdicament cada un número de segons definit per defecte. Aquest número és lo suficientment petit en el temps per enregistrar agents ràpidament però no tant petit com per saturar la xarxa. Un número raonable de segons es trobaria entre 2,5 i 6 segons.

Mètode accept_connexion

En aquest mètode es porta a terme el procediment per acceptar o rebutjar una connexió amb el leader.

Cada agent es connecta com a client TCP al leader del qual ha rebut la petició de registre (missatge REGISTER_AGENTS). El leader és el responsable de determinar si un agent és apte o no per ser participant en la nostra topologia. La decisió de determinar si un agent és vàlid o no és fonamental, doncs pot ocórrer que persones no desitjades vulguin participar en la nostra topologia.

Per tal d'evitar això un leader disposa de certes **mesures de validació** que ens permeten discernir la fiabilitat d'un agent. La primera que es porta a terme és la validació del identificador donat (aquest ha d'estar format per 10 caràcters format per números i lletres). Si l'identificador donat per l'agent no compleix aquest requisit es tanca la connexió.

Amb l'identificador donat també portarem un altre tipus de validació. Aquesta validació es basa en comprovar que l'identificador que ens dóna un agent no formi part ni de la base de dades local del leader ni de la topològica. En aquest cas es considera una **falsificació de la identitat** i es tancarà la connexió.

En la següent imatge es pot veure el fragment de codi de l'acceptació de connexions:

```
data = socket_connexion.recv(8096)
identification = pickle.loads(data)

allowed= action.check_id(identification['id'])

if allowed:
    socket_connexion.setblocking(False)     # TCP Open channel
    socket_connexion.send("ACK".encode())   # Notify agent
    self.connections[identification['id']] = socket_connexion
    role="agent"
    self.num +=1

    if self.num == 1: # backup?
        socket_connexion.send("BACKUP".encode())
        role="backup"

    # Register Agent
    new_agent = Agent(identification['id'],address[0],address[0],identification['port'],False,role,False)
    action.add_agent(identification['id'],address[0],self.ip,identification['port'],role,identification['C
    self.agents.append(new_agent)

else:   # Agent connexion refused
    socket_connexion.close()
```

Fig. 23: Mòdul Agent. Fragment de codi d'acceptació de connexions del leader.

Quan el leader considera que un agent és vàlid passem a agregar-lo a la nostra topologia (figura 24). El leader solament emmagatzema aquella **informació del agent** que és necessària i pot resultar útil per determinar quin agent executa un servei. En aquest projecte s'ha considerat com un factor a tenir en compte per a l'execució de serveis el percentatge de memòria ram i el percentatge de cpu que utilitza un agent.

```
def add_agent(id_agent,ip_agent,ip_leader,port,role,ram,cpu,IoT=None):
    # POst: A new agent is registered.

    if IoT != None:
        devices=[]

        if type(IoT) is dict:
            resources.add_agent(id_agent,ip_agent,role,port,ram,cpu)
            for key,value in IoT.items():
                if key != 'id' and key != 'port' and key != 'RAM' and key != 'CPU':
                    devices.append(key) # IOT FOUND
                    resources.add_devices_agent(id_agent,key,value)           # add
            topology.add_agent(role,id_agent,ip_leader,ip_agent,port,devices) # add
        else:
            id_agent=id_generator()
            resources.add_agent(id_agent,ip_agent,role,port,ram,cpu)
            topology.add_new_agent(role,id_agent,ip_leader,ip_agent,port,IoT)

    else: # Register agent without IoT
        id_agent=id_generator()
        resources.add_agent(id_agent,ip_agent,role,port,ram,cpu)
        topology.add_new_agent(role,id_agent,ip_leader,ip_agent,port)
```

Fig. 24: Mòdul action. Fragment de codi del enregistrament d'agents.

Un leader ha d'estar constantment a l'escolta de la informació o missatges que transmet un agent en un moment donat. Per possibilitar això l'agent leader emmagatzema els **sockets de comunicació** que s'estableixen amb els agents connectats a ell. De la mateixa manera que l'agent cloud els leaders compten amb un diccionari on la clau és l'identificador i el "valor" el socket TCP . Cada socket desat se l'indica que no es bloquegi per a mantenir el canal de comunicació obert.

Quan un agent s'enregistra és decisió del leader determinar si aquest agent pot ser backup o no. Segons el criteri desenvolupat en el projecte el primer agent que s'enregistra és el node backup. Si un node és escollit com a backup se'l notifica pel canal TCP establert.

6.3.1.3 Agents

La posada en marxa dels leaders permet la incorporació dels agents a la topologia. A continuació es descriuen els mètodes principals dels agents per establir la plataforma.

Mètode read_messages_broadcast

L'execució per si sola d'un agent no és rellevant, doncs aquest agent no forma part de cap topologia i no pot executar serveis. Encara i que es poden realitzar peticions des de l'aplicació client aquestes no seran ateses ja que no es compta amb un leader. Per rebre les peticions de registre que envia el leader l'agent s'executa posant-se a l'escolta (mètode bind_agent) dels missatges UDP enviats a l'adreça broadcast.

Si un agent forma part de la **mateixa xarxa** en la que es troba el leader al cap de pocs segons es rep la petició de registre. Quan es localitza un leader l'agent s'intenta connectar a ell com a client TCP mitjançant l'adreça ip del leader (obtinguda a partir del missatge). En la següent imatge es mostra a nivell de codi aquest fet:

```
elif message == b"REGISTER_AGENTS":
    if self.role == "agent":

        if (not self.connected):
            self.ip_leader=str(someone[0])
            action.update_leaderIP(self.ip_leader)
            |
            self.realise_connexion()
            self.connected=True
            self.run_agent()
```

Fig. 25: Mòdul Agent. Fragment de codi on l'agent localitza el leader.

Encara i que l'agent s'enregistri amb un leader sempre seguirà rebent els missatges broadcast de la xarxa. Per evitar que si hi ha dos leaders en la mateixa xarxa l'agent canviï constantment de leader s'ignoren els missatges rebuts quan s'enregistra amb un d'ells mitjançant el booleà Connected com es veu en la figura 25.

Mètode realise_connexion

Un cop l'agent localitza el leader aquest li envia aquella informació (en format diccionari) rellevant per registrar-se: el seu identificador i els seus paràmetres de sistema (memòria ram i cpu). Conjuntament amb aquesta informació s'envia la

informació dels dispositius al càrrec. Aquesta informació és el tipus de dispositiu i l' identificador de cada dispositiu.
Un agent es pot enregistrar sense cap dispositius al seu càrrec ja que pot ser utilitzat per realitzar operacions de processament i computació de dades.

Quan un agent s'enregistra amb un leader executa tres processos en segon pla: l'escolta de missatges del leader, l'actualització de paràmetres i la detecció de caiguda del leader. Aquests tres processos estan relacionats amb el funcionament de la plataforma en execució.

6.3.2 Funcionament de la plataforma en execució

Com succeeix en l'establiment de la plataforma és necessari descriure els mètodes del mòdul Agent per saber com funciona la plataforma en execució. A continuació es descriuen els mètodes més destacats d'aquest mòdul segons el rol dels agents.

6.3.2.1 Cloud

L'únic procés amb el que l'agent cloud efectua en segon pla per al funcionament de la plataforma en execució és l'**escolta de missatges** per part dels leaders. Aquest mètode s'ha declarat però s'ha utilitzat l'API per a la realització de peticions de leaders al cloud ja que aquestes estan relacionades amb la resolució de serveis.

6.3.2.2 Leaders

Els leaders compten amb quatre processos relacionats amb el funcionament de la plataforma: l'escolta de peticions del cloud, l'escolta de peticions per part dels agents, la detecció de caigudes dels agents i l'escolta de missatges via UDP. A continuació es descriuen aquests dos últims (són els més destacats). Com en el cas anterior l'escolta de missatges amb el cloud s'ha declarat però s'ha utilitzat l'API.

Mètode check_agents_alive
Cada leader monitoritza als agents registrats amb ell. Per detectar la caiguda un agent el leader envia periòdicament **missatges de control** (figura 26) al canal TCP establert amb l'agent, per comprovar la seva operativitat.

```
if(int(len(self.agents)) > int(0)):
    for agent in list(self.connections):

        try:
            self.connections[agent].send("CHECK_ALIVE".encode())
        except IOError as e:
            action.print_control_message("[TOPOLOGY]: Agent " + agent +  " is down")

            found=False
            for agent_registered in self.agents:

                if agent_registered.get_id() == agent:
                    if agent_registered.is_backup():
                        action.print_control_message("[TOPOLOGY] Agent backup is down")
                        found=True
                    action.delete_agent(self,agent_registered)
                    self.num -= 1
                    return
            if found:
                action.backup_election(self)
    sleep(2)
```

Fig. 26: Mòdul Agent. Fragment de codi de la detecció de caiguda d'agents.

Si es produeix un error en l'enviament d'aquest missatge serà perquè el canal de comunicació ja no es troba operatiu. Aleshores la informació de l'agent i els seus IoT s'esborra de les bases de dades. Si el node en qüestió és el backup es realitza la selecció d'un altre i si no n'hi ha s'espera al pròxim agent.

Mètode read_messages_broadcast

El mètode que utilitza un agent per rebre peticions també és utilitzat pel leader per rebre informació a través del seu socket UDP. En el cas del leader s'ha utilitzat per rebre la informació dels paràmetres de sistema enviats per l'agent.

Quan el leader rep aquesta informació (en format de diccionari) s'encarrega d'actualitzar la informació d'aquest agent en la base de dades local.

6.3.2.2 Agents

Com abans s'ha comentat hi ha tres mètodes que s'utilitzen per al funcionament de la plataforma en execució. A continuació aquests són descrits.

Mètode read_message

Aquest mètode s'utilitza per rebre la informació dels leaders.

Hi ha quatre tipus de missatges que el leader envia al agent:

- **ACK:** per confirmar que l'agent s'ha agregat a la plataforma.
- **BACKUP**: per assignar el rol de backup al agent.
- **START-COMUNICATION**: per a que aquest agent es connecti amb un altre agent.
- **EXECUTE-SERVICE**: per a que l'agent es posi a l'escolta de connexions realitzades per altres agents.

Aquests dos últims missatges s'utilitzen per a resoldre un servei complexe.

Mètode check_leader_alive

Aquest és el mètode que utilitza un agent per detectar la caiguda d'un leader. Aquesta detecció de la caiguda es realitza de la mateixa forma que efectua el leader, comentada anteriorment en el mètode check_agents_alive.

El procediment a efectuar (fig. 27) per l'agent depèn del seu rol:

- Si l'agent és el backup canvia de rol i s'executa com a leader.
- Si l'agent no és backup es posa a l'escolta de peticions de registre d'altres leaders (Connected=False).

```
if self.backup:
    action.print_control_message("[System] I'm the agent backup.")
    action.print_control_message("[System] Stopping agent tasks...")
    action.print_control_message("[System] Executing agent as a leader...")
    self.socket_leader.close()
    self.socket_broadcast.close()

    self.change_role()                  # stop agent threads
    sleep(5)

    resources.create_topology("leader") # clean database tables and prepare
    topology.delete_agents(self.ip_leader)      # delete agents from topoDB
    topology.add_agent("leader",self.id,"",self.ip,"",self.port)

    self.ip_leader=""
    self.role="leader"
    self.port=self.port+1
    self.realise_binding()              # Sockets tcp/udp binding
    self.run()                          # start leader main threads

    action.print_control_message("[System] Running as Agent Leader.")
else:
    self.connected=False
```

Fig. 27: Mòdul Agent. Fragment de codi de la detecció de la caiguda del leader.

Canvi de rol

El canvi de rol (figura 27) és el **conjunt de procediments** que es porten a terme quan es detecta la caiguda del leader. Els passos que porta a terme un agent per a convertir-se en leader són:

- **L'aturada de tots els processos** que s'executen en segon pla del agent. Per aconseguir aturar els processos d'un agent s'han declarat en el mòdul agent variables booleanes posades a True per cada thread. Cada procés dels agents s'executa sempre i quan aquesta variable no canviï d'estat. En el mètode **change_role** aquestes variables es posen a False i els processos s'aturen progressivament.
- **Eliminar la informació** de la base de dades local: per fer això s'utilitza de nou el mètode create_topology del mòdul resources esmentat anteriorment.
- **L'actualització de la informació topològica.** Per fer-ho s'elimina la informació de tots els agents que estaven enregistrats amb el leader (a través de l'API amb la crida delete_agents). Quan aquests s'agreguin amb el nou leader s'agregaran mitjançant el procediment habitual.
- **L'execució com a leader.** L'agent backup pot executar-se com a leader quan els processos d'agent s'han aturat (mètodes realise_binding i run).

El diagrama de comunicació que es produeix en el canvi de rol és el següent:

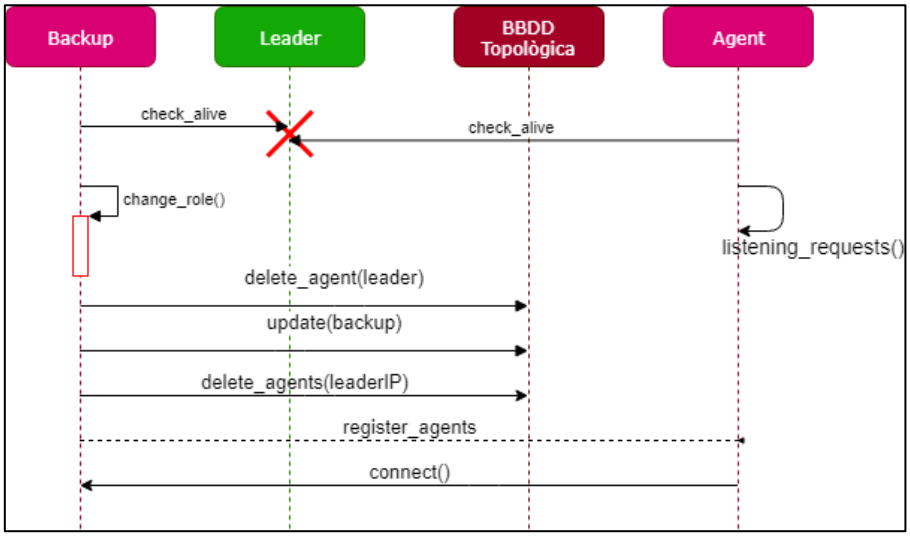


Fig. 28: Diagrama de comunicació que es produeix amb la caiguda d'un leader.

Mètode update_parameters

Cada agent informa periòdicament al seu leader amb la seva informació de sistema (percentatge de cpu i ram ocupat). Aquesta informació (fig. 29) s'entrega via UDP al no ser tanta importància per lliurar (si pel que sigui es perd la informació pel tràfic de xarxa tampoc afectaria massa al rendiment de la plataforma en general).

A diferència de les peticions de registre que són enviades a tota la xarxa aquesta informació és lliurada al leader directament cada 60 segons. Els paràmetres de sistema s'envien cada un cert temps degut a que en pocs segons és difícil que canviïn.

```
ram,cpu=action.get_parameters()
parameters={'RAM':ram,'CPU':cpu}
data=pickle.dumps(parameters)

try:
    self.socket_broadcast.sendto(message.encode(),(self.ip_leader,self.port+1))
    self.socket_broadcast.sendto(data,(self.ip_leader,self.port+1))
```

Fig. 29: Mòdul Agent. Fragment de codi de l'actualització de paràmetres.

6.4 Agent

En aquest apartat es descriuen els mòduls principals d'un agent i els seus corresponents mètodes més destacats a excepció del mòdul Agent que ha sigut descrit anteriorment.

6.4.1 Aplicació client

L'aplicació client està desenvolupada en python3 i mostra un menú amb el llistat de serveis que l'usuari pot sol·licitar amb una descripció de cada servei. Addicionalment es mostren missatges per pantalla segons les tasques que s'està portant a terme el node en qüestió.

Per sol·licitar el servei l'usuari ha d'introduir el nom complet donat o simplement indicar el número del servei. En la següent imatge es veu el menú de l'aplicació:

```
---------------
AGENT CLIENT APP
---------------
List of avaliable SERVICES:
[1][change_color] CHANGE COLOR of the cities' semaphores to Green.
[2][close_barriers] CLOSE the barriers of the city.
[3][change_barriers_error] Request a service that can't be executed.
[4][emergency] Request an ambulance.
[exit] Exit.
---------------
```

Fig. 30: Menú de l'aplicació client

6.4.2 API

Per a desenvolupar l'API d'aquest projecte s'ha utilitzat **Cherrypy**. Cherrypy és un framework d'aplicacions web orientat a objectes que utilitza python com a llenguatge de programació.

Aquesta API s'ha utilitzat com a **eina intermediària** entre mòduls com s'ha explicat en el disseny. Encara i que la classe Agent s'ha utilitzat d'aquesta forma en l'API desenvolupada queda molt més clar aquesta comunicació entre mòduls. Això és degut a que el codi de l'API és molt més reduït. A més, la implementació d'aquesta API ha permès duu a terme una integració amb altres projectes (capítol 8).

Els mètodes d'aquest mòdul poden ser dividits en dos tipus:

- Mètodes relacionats amb la obtenció o modificació de documents de les bases de dades.
- Mètodes que s'utilitzen per a la comunicació entre mòduls, que estan relacionats amb l'execució de serveis.

Els mètodes d'operacions amb les bases de dades són:

- **Register_agent**: per agregar un agent a la topologia. La informació d'entrada és un json.
- **Get_agents:** per obtenir els agents de la base de dades. Aquest mètode és utilitzat per l'agent cloud en la resolució d'un servei. Retorna la direcció ip del leader.
- **Check_agent**: retorna si un agent es troba en la base de dades o no a partir del identificador (string) donat.
- **Delete_agents**: per eliminar un o més agents. Com a entrada pot rebre un identificador o res en concret (aleshores s'eliminen tots els nodes).
- **Update_agent**: per actualitzar la informació d'un agent. Com a entrada rep un json.

Els mètodes relacionats amb els serveis són:

- **Request_Service**: per realitzar una petició de resolució de servei a un node. Com a entrada del mètode es rep un json amb la informació del servei.
- **Execute_Service**: per realitzar una petició d'execució a un node. Com a

entrada es rep un json.

- **Response_service**: per informar dels resultats d'una execució. Com a entrada es rep un json. Aquest mètode s'utilitza per informar tant leader com al client.

Les peticions a una API sempre tenen el següent format:

http://ip_agent:8000/nom_mètode

Cada mètode usat compta amb la directiva **@cherrypy.expose** per a que aquest mètode pugui ser executat per altres aplicacions.

Hi ha mètodes que solament reben un identificador (un string) com a paràmetre però n'hi ha que reben un json. S'utilitza la directiva **@cherrypy.tools.json_in()** per a que un mètode pugui rebre el json correctament.

En la següent imatge es veu l'exemple del mètode response_service:

```
@cherrypy.expose
@cherrypy.tools.json_in() # notification of service results
def response_service(self):
    if cherrypy.request.method == "POST":

        body = cherrypy.request.json

        if self.agent.get_role() == "agent":
            runtime.result_service(body)

        elif self.agent.get_role() == "leader":
            action.print_control_message("[RESOURCES] Updating status of agent located at "
            resources.change_status(cherrypy.request.remote.ip)
```

Fig. 31: Mòdul API. Mètode response_service.

6.4.3 Action

Aquest és un mòdul auxiliar de la classe agent. A continuació es descriuen els mètodes destacats d'aquest mòdul:

Mètodes Add_agents i delete_agents

Aquests dos mètodes s'utilitzen per afegir o eliminar agents de la topologia tant en la base de dades local d'un leader com en la topològica.

Mètode print_control_message

Aquest és el mètode que permet realitzar la **monitorització de nodes**.

Tots els mòduls utilitzen aquest mètode per escriure la informació relacionada amb les tasques del agent. La informació (de tipus string) s'escriu en el fitxer de text que s'envia al administrador.

Mètode backup_election

Aquest mètode és el que utilitza el leader per escollir l'agent backup quan el backup ha caigut o ha abandonat la plataforma. Si un leader compta amb agents enregistrats amb ell s'escull al primer de tots perquè sigui el nou agent backup. Aquesta notificació es realitza via TCP a través del socket entre agent i leader.

Aquests canvi de rol del node es reflecteix tant en la base de dades local com en la topològica. En la següent imatge es pot veure el codi d'aquest mètode:

```
def backup_election(agent):
    # POST: TRUE. The first agent connected to a leader will be the new agent backup

    list_agents= agent.get_list_agents()
    connexions = agent.get_connexions()

    if len(list_agents) > 0:

        print_control_message("[TOPOLOGY] Choosing a new agent backup...")
        id_agent = list_agents[0].get_id()              # THE NEW AGENT BACKUP WILL BE THE F
        list_agents[0].set_role("backup")
        connexions[id_agent].send("BACKUP".encode('utf-8'))    # nofitication agent
        resources.update_role(id_agent)                 # update database local
        topology.update_agent(agent) # update global database
        print_control_message("[TOPOLOGY] Agent " + id_agent + " is the new agent backup." )

    else:
       print_control_message("[TOPOLOGY] Waiting for a new agent to be backup...")
```

Fig 32: Mòdul Action. Mètode d'elecció del node backup.

6.4.4 Service Execution

En aquest mòdul hi consten totes les crides a l'API relacionades amb els serveis abans esmentades. Exceptuant aquestes crides solament hi consta un mètode en aquest mòdul, el mètode attend_service.

Mètode Attend_Service

Aquest mètode és el que utilitzen els agents per **gestionar les peticions de servei**. En aquest mètode està implementat com resoldre un servei. Aquest mètode és descrit en el desenvolupament tècnic dels serveis (apartat 6.5 Serveis).

6.4.5 Topology

En aquest mòdul hi consten tots els mètodes relacionats amb les operacions amb la base de dades topològica i de serveis mitjançant les crides abans esmentades en el mòdul API.

6.4.6 Resources

En aquest mòdul hi consten els mètodes que s'utilitzen per interaccionar amb la base de dades local d'un agent o leader. Aquest mòdul pot ser modificat o accedit des dels mòduls Agent i Action (per agregar o eliminar agents), pel service execution (per obtenir informació d'agents) i per l'API (per actualitzar l'estat d'un agent).

Mètode realise_connexion

Aquest mètode és usat per tots els mètodes d'aquest mòdul. En aquesta funció es realitza la connexió amb la base de dades a partir del usuari, contrasenya, host (localhost) i el nom de la base de dades (resources).

6.4.7 Runtime

En aquest mòdul un agent porta a terme l'execució d'un servei a partir de tres mètodes principals:

Mètode dowload_file

Aquest mètode s'utilitza per descarregar un fitxer d'un servidor remot. Aquesta descàrrega es realitza a partir de la informació donada per realitzar una connexió: host, usuari, contrasenya, directori remot.

Execute_service

Aquest és el mètode on es produeix l'execució d'un servei. L'execució d'un servei es produeix quan:

- Tots els **fitxers** necessaris són descarregats.
- S'adjunta la **versió del programa** i els **paràmetres** en una llista.
- L'**execució** es realitza de forma que l'agent s'espera al codi que retorna l'execució. Si l'execució retorna el codi 0 significarà que l'execució s'ha portat a terme correctament. Altrament es considera que ha ocorregut un error.

Send_results

En aquest mètode s'obtenen els resultats d'una execució (a partir del fitxer results.json). Un cop s'ha realitzat la lectura dels resultats aquests s'envien als agents enregistrats amb l'agent coordinador mitjançant els sockets TCP.

6.5 Serveis

El desenvolupament tècnic dels serveis és clau per aconseguir una plataforma ràpida i eficient. Per això a continuació s'expliquen com s'han desenvolupat els punts clau de tot el procés de vida d'un servei. Aquest comprèn des de que és demanat des del client fins a la notificació de resultats.

6.5.1 Petició de servei

Els agents i leaders són els nodes de la topologia que realitzen peticions de servei a través d'una aplicació client.

Quan el client realitza una petició de servei el que es produeix és una crida des de l'aplicació client a l'agent mitjançant l'API d'aquest. Si el servei sol·licitat requereix de paràmetres (coordenades, dades introduïdes per l'usuari...) aquesta informació és adjuntada a la petició com es veu en la següent imatge:

```
elif command[0] == "change_color":
    service="CHANGE_COLOR"

    color = "Green"
    latitude=round(random.uniform(0,10),2)
    longitude=round(random.uniform(0,10),2)
    list_values=[color,latitude,longitude]

    service= {"service_id":service,'params':[color,latitude,longitude]}
    service_execution.request_service(ip_address,service)
```

Fig. 33: Mòdul App_agent. Petició del servei canvi de color.

6.5.2 Resolució d'un servei pel leader

La petició de servei enviada pel client és rebuda per l'agent (mòdul service_execution). Cada agent sempre delega la petició de servei al seu leader ja que es responsabilitat d'aquest node la seva resolució.

Quan la petició de servei arriba al leader aquesta passa a ser resolta . Primerament el que fa el leader és obtenir tota la informació del servei (figura 34) demanat a partir del seu identificador accedint al catàleg de serveis des de la pròpia API.

```
select={'service_id':serv["service_id"]}
service_list=[]
for service_mongo in self.services.find(select):  # find service
    service_list.append(service_mongo)

service_json = json.dumps(service_list, default=json_util.default)
service= json.loads(service_json)
service[0]['ip_agent_request']=cherrypy.request.remote.ip
service_execution.attend_service(self.agent,cherrypy.request.remote.ip,service[0],serv)
```

Fig. 34: Mòdul API. Gestió de la petició d'un servei pel leader.

Prèviament de la pròpia resolució el leader s'encarrega d'adjuntar a la informació del servei els **paràmetres d'execució** (llista) enviats pel client i la **informació per accedir al servidor** remot per la descàrrega del codi. Aquesta informació (definida en variables) és una llista on consta el host, l'usuari, contrasenya i directori remot.

6.5.2.1 Resolució d'un servei simple

La resolució d'un servei simple es basa en trobar els agents que compten amb els dispositius requerits. Si un servei no requereix de dispositius es realitzarà una cerca del node amb menor consum de ram i cpu (servei computacional). De la base de dades s'obté una llista amb les adreces ip dels nodes que compten amb els IoT.

A cada node de la llista se li donarà l'ordre d'execució amb la crida execute_service a través de l'API de cada node. En la següent imatge es mostra el fragment de codi corresponent a la resolució d'un servei que requereix de dispositius:

```
if 'IoT' in service: # look for IOT owners
    ips_agent_target=[]
    list_iot=service['IoT']

    ips_agent_target=resources.find_type(list_iot,ip_agent) # LIst of agents that could

    if len(ips_agent_target) > 0:
        if len(ips_agent_target) == 1:
            service['Notification']=True
        list_involved=[ip_agent]

        for agent_service in ips_agent_target: # SERVICE EXECUTION ORDER FOREACH AGENT
            list_involved.append(agent_service)
            execute_service(agent_service,service)

        service['Output']=list_involved # Total of agents involved in service
```

Fig. 35: Mòdul Service Execution. Fragment de codi de la resolució d'un servei simple.

Per cada agent obtingut el leader realitza la petició d'execució (crida de l'API execute_service).

A continuació es mostra el diagrama de comunicació d'un servei simple:

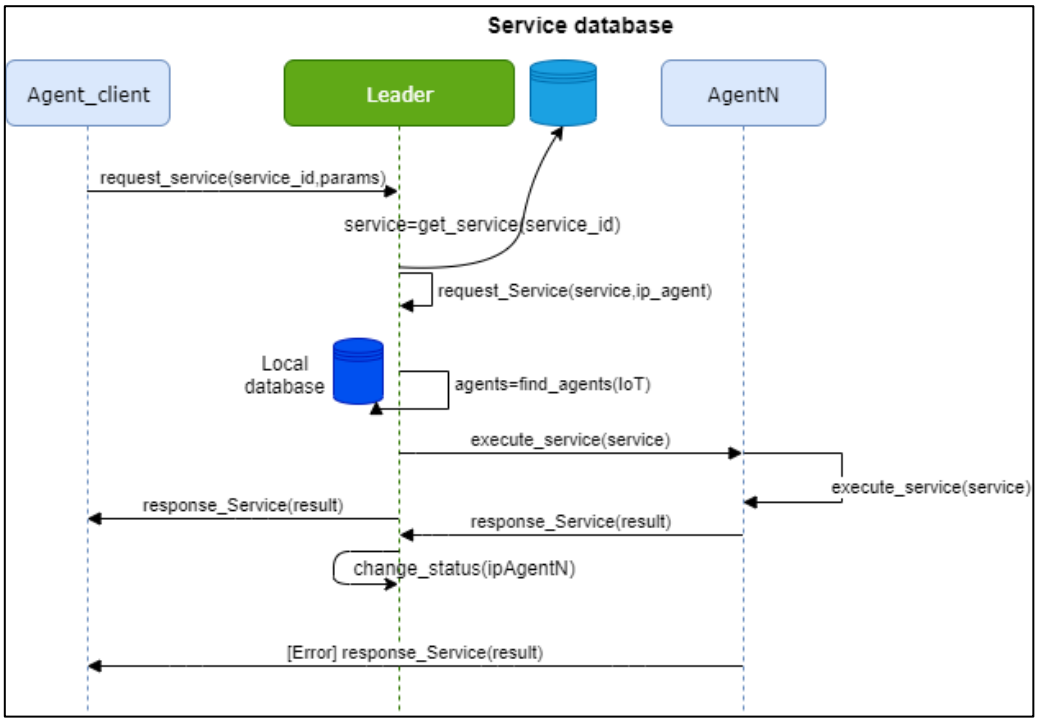


Fig. 36: Diagrama de comunicació d'un servei simple.

6.5.2.2 Resolució de servei dirigit

Aquest tipus de servei es resolen més ràpidament degut a que no cal cercar cap agent. En els serveis dirigits es **notifica directament als agents** (mitjançant la crida execute_service de l'API) que es troben indicats en la definició del servei. En aquests serveis es troba definit el camp IP_AGENTS on el valor és la llista amb les adreces ip dels nodes que poden executar el servei.

6.5.2.3 Resolució d'un servei complexe

La resolució d'un servei complexe comença per trobar **l'agent coordinador** del servei. L'agent coordinador és escollit pel menor percentatge de cpu i ram utilitzat. Com que aquest agent s'encarregarà de comunicar-se amb altres agents i de la notificació de resultats aquest ha de pertànyer a un agent amb cert poder computacional.

L'obtenció de la informació de l'agent coordinador s'obté mitjançant el mètode **agent_election** del mòdul resources.
A continuació es mostra el fragment de codi corresponent:

```
id_exec,ip_address,port = resources.agent_election(ip_agent,service['resources'])

if id_exec != "": # AGENT IN CHARGE FOUND

    action.print_control_message("[SERVICE EXECUTION] Agent located at "+ ip_address +" with id "
    resources.update_status(id_exec)
    service['ip_agent_request']=ip_agent

    connexions = agent.get_connexions()                    # Dict with TCP sockets, KEY: AGENT I
    connexions[id_exec].send("EXECUTE-SERVICE".encode())   # NOtify agent in charge
    data=pickle.dumps(service)
    connexions[id_exec].send(data)

    for sub_service in service['Dependencies']:
        data={'service_id':sub_service,'ip_address':ip_address,'port':port,'id_exec':id_exec}
        request_service(agent.get_ip(),data)   # service requests send to the self agent
```

Fig. 37: Mòdul service execution. Fragment de codi de la resolució d'un servei complexe.

Quan aquest agent és escollit se l'hi notifica de l'execució del servei principal mitjançant el socket TCP enviant el missatge EXECUTE-SERVICE. Aleshores l'agent coordinador es prepara (crea un socket TCP) per rebre connexions d'altres agents.

Durant l'execució del servei principal el leader s'encarrega de resoldre els serveis dependents del primer. Per resoldre aquests serveis el leader es realitza peticions de resolució a si mateix per cadascun d'aquest serveis (és a dir que es duu a terme una recursivitat amb la crida **request_service**). El leader al rebre una petició de resolució de si mateix busca la informació d'aquest servei i determina de quin tipus es tracta.

Cada servei depenent del primer es resol de la mateixa manera que un servei simple. La principal diferència recau en que el leader informa als agents escollits de que es connectin al agent coordinador. Aquesta notificació es realitza mitjançant el missatge START-COMUNICATION. A la informació del servei s'adjunta l'adreça ip i port de l'agent coordinador. Aquest procediment es pot veure en la imatge 38.

```
ids_agent_target=resources.find_type_id(service['IoT'],ip_agent,arguments['id_exec'])
connexions = agent.get_connexions()                    # Dict with TCP sockets, KEY:

if len(ids_agent_target) > 0:

    for agent_service in ids_agent_target:
        resources.update_status(agent_service)
        connexions[agent_service].send("START_COMUNICATION".encode())

        service['ip_target']=arguments['ip_address']  # AGENT IN CHARGE LOCATION
        service['port']=arguments['port']

        data=pickle.dumps(service)
        connexions[agent_service].send(data)
else:
    # NO more agents avaliables
    delegate_cloud(service,ip_agent)
```

Fig. 38: Mòdul service execution. Fragment de codi de la resolució dels serveis dependents.

Si algun d'aquests serveis no ha pogut ser resolt pel leader aquest delega la petició al agent cloud (mètode **delegate_cloud**) perquè trobi agents d'una altra zona per a la seva resolució.

En la següent imatge es mostra el diagrama de comunicació d'un servei complexe:

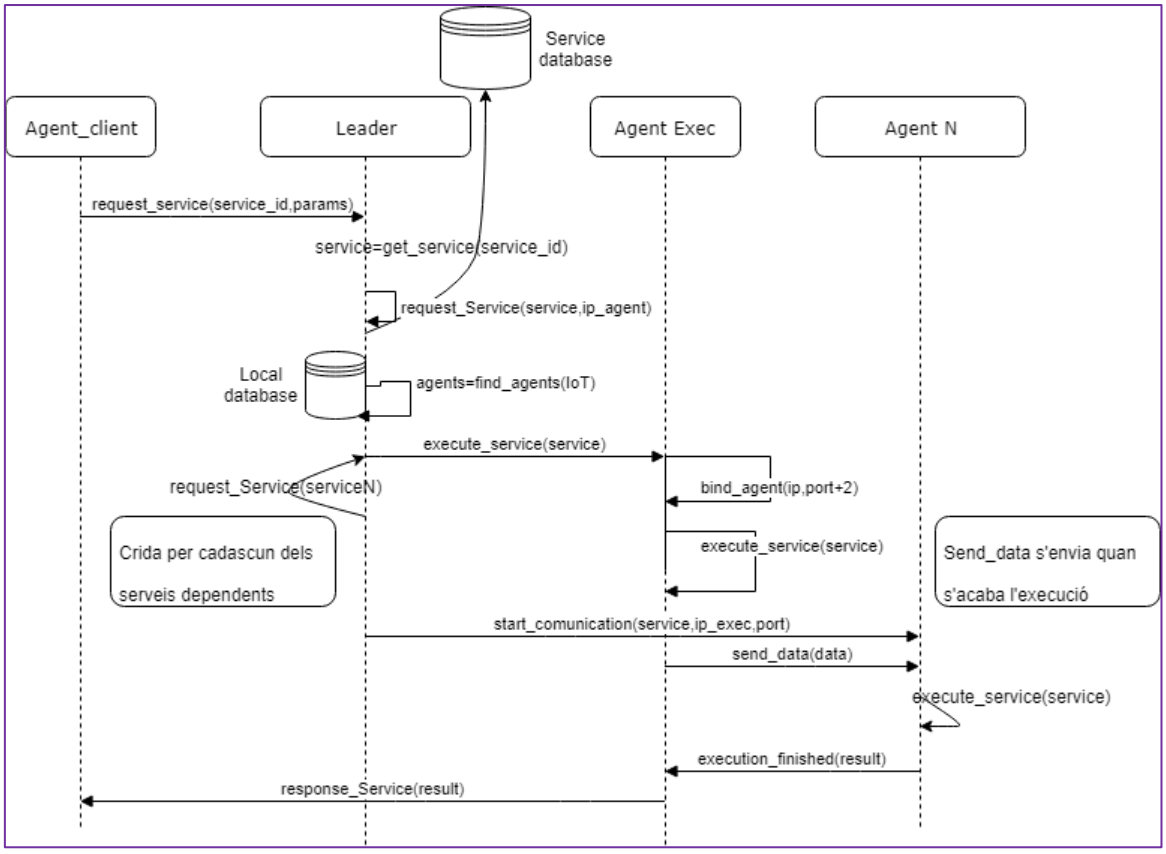


Fig. 39: Diagrama de comunicació d'un servei complexe.

6.5.3 Resolució de servei per l'agent cloud

L'agent cloud duu a terme la resolució del servei quan un leader no comptava amb els agents suficients. A diferència del leader aquest node no es preocupa del tipus de servei sol·licitat ni de buscar la informació d'aquest. Com que la informació del servei la envia el leader l'agent cloud solament s'encarrega de si hi ha un agent que compta amb els dispositius requerits en la base de dades topològica. En la següent imatge es mostra el fragment del codi corresponent a la resolució d'un servei:

```
ip_leader = topology.find_agent_owner(service['IoT'])

if ip_leader != None: # NOTIFY LEADER
    action.print_control_message("[SERVICE EXECUTION] Agent with IoT requested found.")
    action.print_control_message("[SERVICE EXECUTION] Delegating order to Agent leader located at

    full_url= url + str(ip_leader) + ":" + str(API_port) + "/request_service"  # API call
    #service['params']=arguments
    result = requests.post(full_url,json=service) # Order to agent
else:
    action.print_control_message("[SERVICE EXECUTION][ERROR]: Service " + service['service_id']  +
    #ERROR NOTIFICATION
    data={'result_code':"ERROR",'service_id':service['service_id'],'Description':"Service " + serv
    #notify_error(service['ip_agent_request'],service,error1)
    response_service(service['ip_agent_request'],data)
```

Fig. 40: Mòdul service execution. Resolució de serveis per l'agent cloud.

Quan l'agent cloud troba un agent d'una altra zona (s'obté una adreça ip d'un leader) s'encarrega de delegar la petició (request_service a l'API del node). a aquest node per a que ell s'encarregui de resoldre el servei. L'agent cloud solament s'encarrega de verificar la presència d'agents que poden resoldre el servei en una altra zona. La resolució del servei, la cerca dels agents (ja que pot haver més d'un) i la demanda de l'ordre d'execució són delegades al leader en qüestió.

Si l'agent cloud no troba cap agent que pugui resoldre el servei demanat aquest **notifica directament al client** del tipus d'error succeït.

6.5.4 Connexió entre agents

En els serveis complexes hi ha un pas previ a l'execució del propi servei. Aquest pas varia segons si l'agent executa el servei principal o un de dependent.

L'agent coordinador prèviament a l'execució es posa a l'escolta de connexions TCP

(en el número de port +2 l'agent crea un altre socket TCP). Per fer-ho aquest posa en marxa els processos d'acceptació de connexions i d'escolta de missatges dels agents. Aquests dos processos estan implementats de la mateixa manera que els processos de l'agent cloud.

Com que aquesta escolta es realitza prèviament a l'execució aquest agent pot executar el servei assignat al mateix temps que accepta noves connexions d'altres agents paral·lelament.

Els agent que reben el missatge START-COMUNICATION es connecten amb l'agent coordinador i posen en marxa el procés d'escolta de missatges amb quest agent.

6.5.5 Execució de servei

Quan un agent rep l'ordre d'execució d'un servei per part del leader o d'un altre agent es disposa a procedir amb l'execució (mòdul runtime).
Per executar un servei un agent es descarrega els codis (codi principal i dependents)mitjançant el mètode download_file. Aquesta descàrrega es fa amb la informació proporcionada pel leader del servidor SFTP del cloud.

L'execució del servei (figura 40) es realitza amb una llista dels arguments corresponents. Aquests arguments poden provenir del agent que sol·licita el servei, d'un resultat d'execució d'un altre agent o poden estar definits en la definició del servei en si.

```
execution_process = subprocess.Popen(list_params,stdout =subprocess.PIPE) #Execution though a subprocess
streamdata= execution_process.communicate()[0]

if execution_process.returncode == 0:
    action.print_control_message("[RUNTIME] Execution of service completed.")
    if 'Notification' in service:
        if service['Notification'] == True:
            data={'result_code':"SUCCESS",'service_id':service['service_id'],'Description': "The executio
        service_execution.response_service(service['ip_agent_request'],data)
else:
    data={'result_code':"ERROR",'service_id':service['service_id'],'Description': "The execution of the s
    service_execution.response_service(service['ip_agent_request'],data)
    action.print_control_message("[RUNTIME] Execution of service failed.")
```

Fig. 41: Mòdul Runtime Execució del servei amb els paràmetres del client.

L'agent s'espera al codi d'execució per a determinar l'èxit o fracàs de l'execució. Per fer-ho s'utilitza una *pipe* per esperar-se al resultat de l'execució del procés.

Quan l'execució finalitza si l'agent ha de notificar els resultats ho fa mitjançant la crida a l'API del client response_service amb el resultat i una descripció del succeït. Si el servei que executa un agent és el servei principal d'un servei complexe aquest node s'encarrega d'enviar els resultats de l'execució a tots els agents connectats a ell. Els resultats es troben en **format json** en el fitxer generat "result.json" i s'utilitza el mètode **send_results** per enviar els resultats.

6.5.6 Finalització d'un servei

Un servei es dóna per finalitzat quan el resultat ha sigut notificat al client i tots els agents que han participat poden tornar a executar un altre servei.

La notificació de resultats es produeix tant al leader (per actualitzat la disponibilitat d'un agent) com al client. Aquesta notificació es realitza mitjançant la crida **response_service** a l'API del node en qüestió a partir de la seva adreça ip.

En cas de que es produeixi un error en la pròpia execució **cada agent informa del error al client**. Aquests errors poden ser deguts a un error sintàctic del codi o perquè el servei en si s'ha executat incorrectament per com ha finalitzat.

Solament en els serveis complexes és necessari tancar les comunicacions establertes entre agents. Aquestes comunicacions es tanquen de la mateixa forma que un agent canvia de rol: aturant els processos adients i tancant el socket TCP utilitzat.

6.6 Front-end

En aquest apartat es descriu el desenvolupament tècnic del front-end.

6.6.1 Informació topològica

En l'apartat Topology del front-end l'administrador pot veure la informació global de la topologia. Per a que l'administrador visualitzi correctament la informació topològica des del front-end es realitza una connexió al MongoDB de l'agent cloud.

6.6.2 Informació d'agents

Des del front-end es poder accedir a la informació de cadascun dels agents. Per poder-ho fer des de el frontend es realitza una **connexió a la base de dades local**

MySQL del agent en qüestió. Per a la connexió s'utilitza les mateixes dades que els s'utilitzen en el mòdul resources d'un agent (mètode realise_connexion).
Si l'agent seleccionat és un leader es cercarà per la informació dels agents que té assignats (taula agents). En canvi si l'agent és un agent qualsevol es mostrarà els tipus d'IoT (taules semaphore, ambulance i barrier) que té assignats.

6.6.3 Informació d'execució

Cada agent envia la seva informació a la màquina de l'administrador. Per fer-ho cada node genera un fitxer de text que es sobreescriu a mesura que l'agent realitza diverses tasques.

Cada agent s'encarrega de pujar el fitxer de text generat al servidor SFTP de l'administrador mitjançant l'execució del programa **upload.py**. Si la màquina del administrador es troba operativa desarà el seu fitxer amb la informació d'execució al directori remot indicat en aquest programa.

Cada 2s es carrega de nou el fitxer en qüestió. Aquesta actualització periòdica permet mantenir informada a la persona que utilitza al front-end sense que aquest tingui que refrescar la pàgina. En la següent imatge es mostra el fragment de codi adient:

```
<script>
    $(function(){
        var id = document.getElementById("id_agent").value;
        var url='load.php?id=' + id;

        // load every 2 seconds
        setInterval(function(){
            $.get(url, function(response){

                document.getElementById("data-text").innerText=response;
            } );
            }, 2000);
    });
</script>
```

Fig. 42: Fragment de codi del front-end de l'actualització periòdica de la informació d'execució d'un agent.

6.6.4 Serveis

Per mostrar la informació del catàleg de serveis i la pròpia de cada servei es realitza una connexió al MongoDB de l'agent cloud, a la col·lecció de serveis.

7. TESTEIG DEL PROJECTE

Durant el desenvolupament d'aquest projecte s'han realitzat diverses proves per comprovar el funcionament de la plataforma dissenyada.

En aquest apartat es detallen quines proves s'han realitzat per comprovar el funcionament del desenvolupament tècnic realitzat.

Les proves es basen en testejar el funcionament de cada node (i com a resultat el de la plataforma) i el de comprovar el funcionament dels punts clau del projecte com són el funcionament de la plataforma en execució i l'execució de serveis.

7.1 Test de funcionament de l'agent cloud

Si l'agent cloud s'executa correctament s'hauria de poder:

- Visualitzar el llistat de serveis i la seva informació des del front-end.
- Veure en el servidor SFTP els fitxers de tots els serveis.

Per comprovar que l'agent Cloud s'ha executat correctament primerament caldrà comprovar que en el catàleg de serveis hi ha tota la informació dels serveis definits en l'script d'inicialització.

Per fer-ho hauríem d'accedir al front-end. En l'apartat "Services" i ens hauria d'aparèixer el llistat de serveis que ha carregat com en la següent imatge:

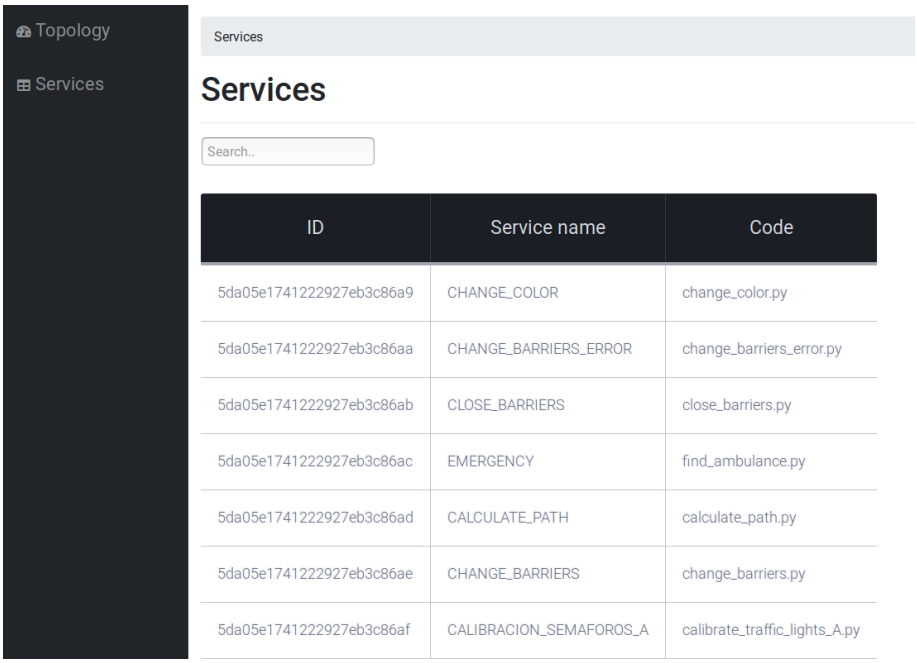


Fig. 43. Visualització de l'apartat de serveis del front-end.

Si es clica sobre un servei qualsevol se'ns hauria de mostrar tota la informació d'aquest, segons està definit en el programa d'inicialització com es veu en la següent imatge:

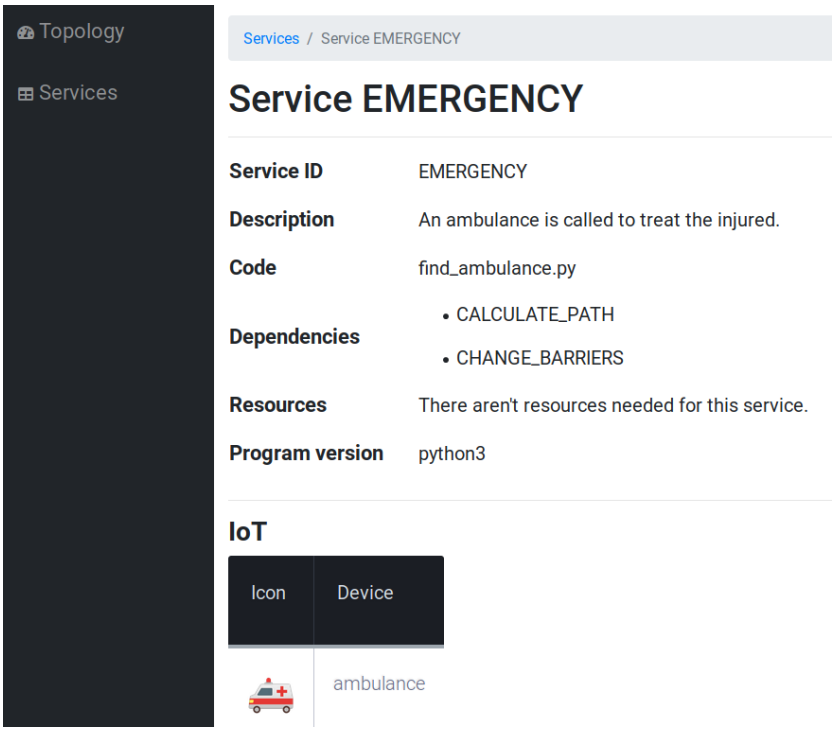


Fig. 44: Informació del servei d'emergències.

Per acabar de comprovar que aquest agent s'ha executat correctament cal comprovar que en el directori remot del servidor SFTP es troben tots els fitxers dels serveis definits. Per fer-ho es suficient en veure el contingut del directori remot de la màquina de l'agent cloud i veure que es troben els fitxers.

7.2 Test de funcionament dels leaders

Els objectius d'aquest test són:

- Verificar l'enregistrament dels leaders amb el cloud.
- Verificar el funcionament de l'apartat topològic del front-end.

Primerament cal comprovar que un leader s'executa adequadament. En el moment que aquest s'executa s'hauria de connectar amb l'agent cloud automàticament.
Si el leader es connecta amb el cloud faltarà per comprovar que aquest s'ha enregistrat correctament en la base de dades topològica.

Des del front-end es pot comprovar quins nodes estan enregistrats en la base de dades topològica accedint a l'apartat Topology.
En la següent imatge es mostra l'apartat Topology del front-end on es pot veure com el leader i un altre node s'han agregat a la plataforma:

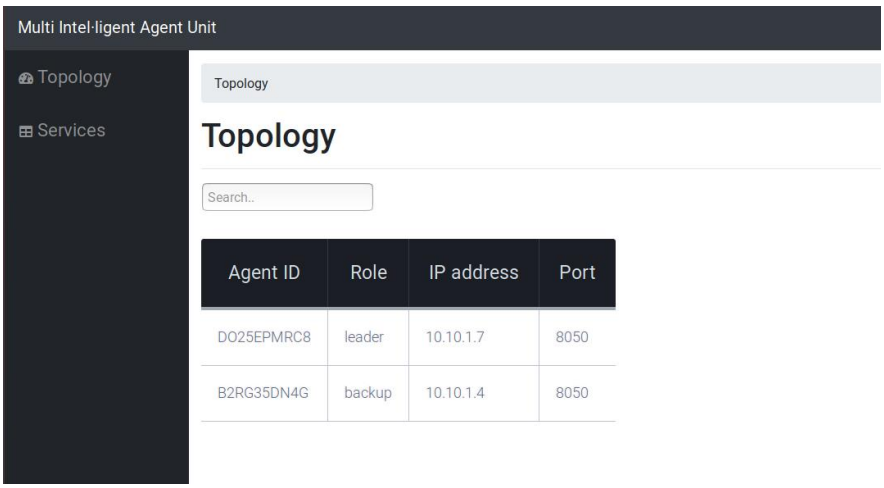


Fig. 45: Visualització de l'apartat Topology del front-end.

7.3 Test de funcionament dels agents

Els objectius d'aquest test són:

- Verificar l'enregistrament correcte d'agents amb leaders.
- Verificar la correcta generació d'IoT.
- Verificar que des del front-end es pot veure la informació d'agents i leaders.

Si l'agent cloud i un leader estan en marxa es pot procedir a executar els agents. Per fer-ho cal executar l'agent en la mateixa subxarxa. Si tot ha anat bé al cap d'uns segons aquest hauria d'estar enregistrat amb el leader (en la pantalla de l'agent es mostra un missatge de que l'enregistrament ha funcionat). Com que el primer agent és el backup escollit pel leader ens hauria d'aparèixer per pantalla que l'agent d'aquesta zona és el backup.

Per verificar que el leader ha enregistrat aquest agent en la topologia cal comprovar-des del front-end. Al clicar sobre el leader en l'apartat "Area Topology" s'haurien de mostrar els agents que s'han registrat amb ell com es veu en la següent imatge:

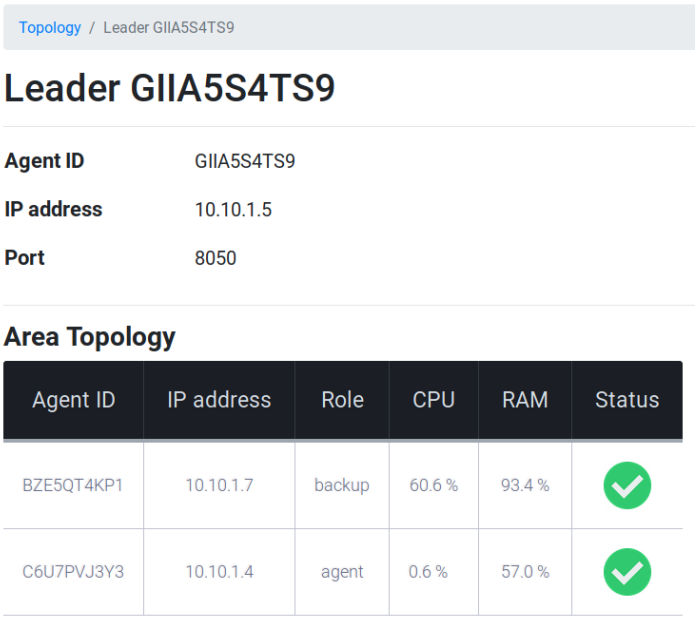


Fig. 46: Visualització de la informació d'un leader des del front-end.

Per veure la informació d'un agent solament cal clicar en aquest des de la informació del leader o des de la pantalla principal de l'apartat Topology. Dels agents a part de mostrar la seva informació bàsica (direcció ip, direcció ip del seu leader...) també se'ns mostra la informació dels seus dispositius.

A continuació es mostra la informació que proporciona el front-end dels dispositius d'un agent:

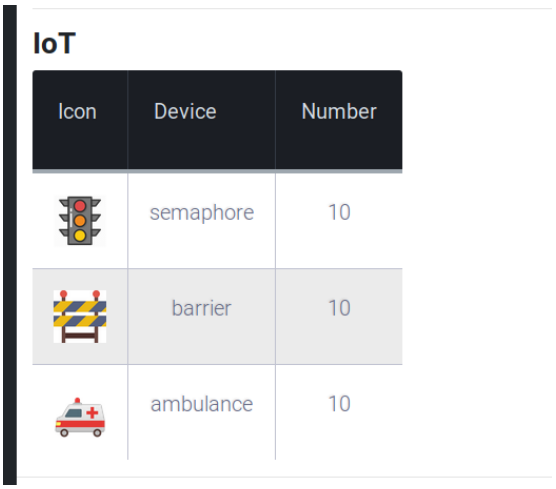


Fig. 47: Visualització de la informació dels dispositius d'un agent des del front-end.

7.4 Test de funcionament del sistema anti-caigudes

A continuació s'expliquen com es pot comprovar que el sistema anti-caigudes desenvolupat funciona correctament.

7.4.1 Detecció caiguda d'un agent

Per a detectar la caiguda d'un agent un leader és suficient amb tenir un agent enregistrat amb un leader. En aquesta prova a part de comprovar això també es comprovarà que el **leader escull un altre agent** com a backup. Per realitzar aquestes proves s'ha de comptar amb un leader amb almenys dos agents enregistrats a ell.

Des del agent backup tancarem el terminal forçosament per tal de que les comunicacions no es tanquin correctament i el leader no sigui notificat. Ens hauria d'aparèixer el següent missatge per pantalla:

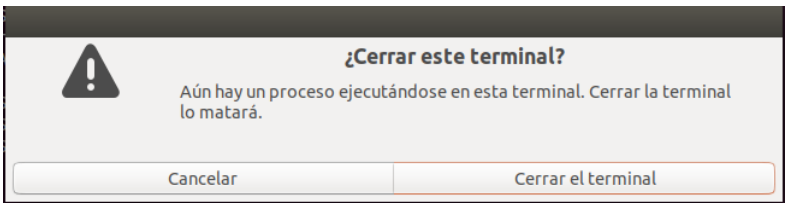


Fig. 48: Notificació de tancament d'aplicació forçós.

El leader mitjançant el sistema de detecció detecta la caiguda i comprova si té més agents enregistrats amb ell. Com que el leader té un altre enregistrat el notifica que serà el backup amb el missatge "BACKUP". Quan l'agent rep la ordre es veu per pantalla el missatge adient. Tot aquest procés es pot veure des de la pantalla de cada agent.

7.4.2 Detecció caiguda d'un leader i canvi de rol

Aquesta prova s'ha de realitzar amb un leader i dos agents enregistrats amb ell.
D'igual manera com es feia en l'apartat anterior cal forçar el tancament forçós de l'aplicació d'un agent, en aquest cas del leader.
La caiguda del leader ocasiona que el backup actuï i s'executi com a leader aturant els processos oportuns.

Quan el leader cau els altres **agents tornen al seu estat inicial** i es posen a l'escolta de peticions de registre.

En el moment en el que l'agent backup ha finalitzat el seu canvi de rol aquest comença a realitzar les funcions del leader, començant per la captació d'agents. Quan els agents de la zona reben la petició de registre d'aquest agent s'enregistren amb ell mitjançant el procediment habitual.

Tots aquests fets es poden veure des del front-end d'administració en l'apartat informació d'execució de cada node.

7.5 Test de la monitorització de la plataforma

Per veure que el està executant un agent cal accedir al front-end. En l'apartat "Agent Execution" de la d'informació de qualsevol agent es pot veure en temps real les tasques que està realitzant en el moment actual.
Aquesta prova de funcionament es pot veure en les següents proves ja que les imatges de les tasques que porta a terme un agent s'han extret del front-end.

7.6 Test de l'execució de serveis

En aquest apartat s'expliquen totes les proves que s'han portat a terme per verificar que un servei és executat correctament des de que es sol·licita fins que es notifica al client amb el resultat.
Els serveis s'ha implementat de forma que realitzen operacions molt simples (contra una base de dades o càlculs aleatòris) però permeten testejar el funcionament de la plataforma.
Els següents implementats són els següents:

- Servei canvi de color semàfor (simple).
- Servei canvi de barreres erroni (simple). Aquest servei retorna error.
- Servei tancament barreres (dirigit).
- Servei d'emergències (complexe). Aquest servei està format per dos serveis dependents: el càlcul de ruta i el canvi de barreres.

7.6.1 Petició de servei

Per comprovar que la petició de servei feta per un agent a un leader funciona efectuarem la petició a d'un servei (introduint el nom del servei o el número en qüestió) des de l'aplicació client de l'agent.
La recepció de la petició s'hauria de veure en la pantalla del leader.

7.6.2 Resolució d'un servei pel leader

Per a realitzar les proves de resolució serà suficient en comprovar la resolució que el leader porta a terme en un servei simple i un de complexe.

7.6.2.1 Servei simple

Per a resoldre un servei simple cal comptar amb almenys dos agents enregistrats amb un leader. Des d'un d'aquests es realitzarà la petició d'un servei simple, per exemple el servei canvi de color.

El leader rep la petició, determina el tipus de servei i procedeix a resoldre'l. Realitza la cerca d'agents en la seva base de dades local a partir dels IoT. En el servei canvi de color solament es requereixen els semàfors. D'aquesta cerca s'obté una llista d'adreces IP. Seguidament el leader dóna l'ordre d'execució a cadascun d'ells. Quan l'última execució finalitzi el leader s'encarregarà de notificar els resultats.

Tot el procediment esmentat es veu en la següent imatge:

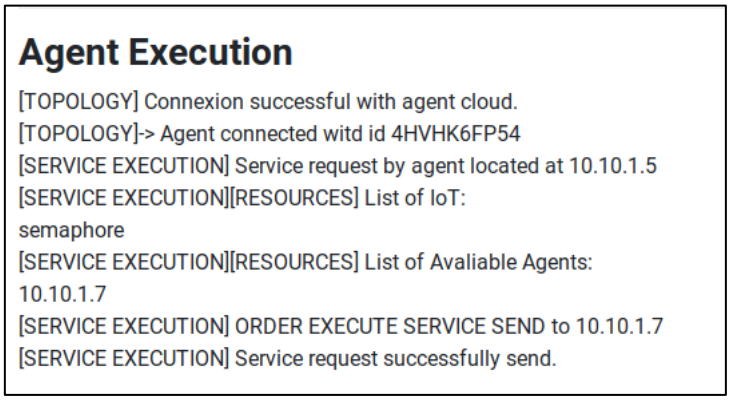


Fig. 49: Resolució del servei canvi de color.

7.6.2.2 Servei complexe

Per resoldre el servei d'emergències desenvolupat en aquest projecte s'ha de comptar amb tres agents enregistrats amb un leader.

El leader rep la petició de servei i procedeix a resoldre-la. Quan determina que el servei és complexe (fig. 50) primerament s'encarrega de buscar al **agent coordinador**. Un cop escollit aquest node el leader es realitza dues peticions de resolució de servei a si mateix: el càlcul de ruta i el canvi de barreres de la ciutat.

```
[SERVICE EXECUTION] Service request by agent located at 10.10.1.7
[SERVICE EXECUTION] Building cluster of agents...
[SERVICE EXECUTION] Agent located at 10.10.1.6 with id 7INNWMGS8F is in charge of the execution
[SERVICE EXECUTION] Service request by agent located at 10.10.1.5
[SERVICE EXECUTION]: EXECUTING SERVICE CALCULATE_PATH WITH DEPENDENCIES
[SERVICE EXECUTION] Service request successfully send.
[SERVICE EXECUTION] Service request by agent located at 10.10.1.5
[SERVICE EXECUTION]: EXECUTING SERVICE CHANGE_BARRIERS WITH DEPENDENCIES
[SERVICE EXECUTION] Service request successfully send.
```

Fig. 50: Resolució del servei d'emergències.

Per cada servei el leader busca els agents que poden executar el servei d'igual forma que en la resolució del servei simple. El servei de canvi de ruta és un servei de computació que no requereix d'IoT i el canvi de barreres requereix les barreres.

Seguidament, l'**agent coordinador** es posa a l'escolta de connexions d'altres agents i comença a executar el servei amb els paràmetres d'execució aportats pel client. Quan finalitza l'execució envia els resultats que necessiten la resta d'agents que s'han enregistrat amb ell. En la següent imatge es pot veure aquests fets:

```
[RUNTIME]: Order execute-service received
[RUNTIME] File already exists, executing service...
[RUNTIME] Executing service find_ambulance.py with execution parameters.
[RUNTIME] New agent connected with id GH9NEYL5F1
[RUNTIME] New agent connected with id S940SLSHC1
[RUNTIME] Execution of service completed.
[RUNTIME] Sending result's file to known-agents...
[RUNTIME] Results sent.
[RUNTIME] Stopping service threads...
```

Fig. 51: Execució de servei complexe per l'agent coordinador.

Els altres agents primerament es connecten al coordinador i quan reben els resultats de l'agent coordinador procedeixen amb l'execució (figura 52). En aquest cas un agent s'encarrega de calcular la ruta de l'ambulància al client i l'altre de canviar les barreres de la ciutat mitjançant una operació a la base de dades local.

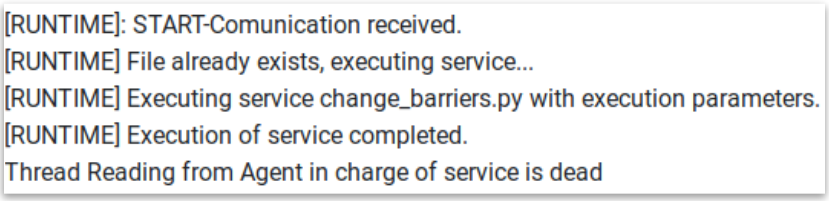


Fig. 52: Execució del servei secundari canvi de barreres.

Quan cada agent tanca les comunicacions notifica al coordinador dels resultats. Aquest s'encarrega d'enviar els resultats al client i de tancar les comunicacions. El leader rep la confirmació de que cada agent ha finalitzat d'executar el servei .

7.6.3 Resolució per l'agent cloud

En aquesta prova es verificarà com l'agent cloud ajuda en la resolució dels serveis als leaders. Per portar-la a terme cal comptar amb dos leaders i dos agents que estiguin registrats en leaders diferents.

Des d'un dels agents es realitzarà la petició d'un servei. Durant el procés de resolució el leader determinarà que no hi haurà cap altre agent disponible. Aleshores, delega en l'agent cloud la resolució d'aquest.

L'agent cloud rep la petició del leader i busca algun agent en la base de dades topològica que compti amb el tipus d'IoT requerits. Quan el troba agafa la direcció IP del altre leader i delega en aquest la resolució (figura 53).

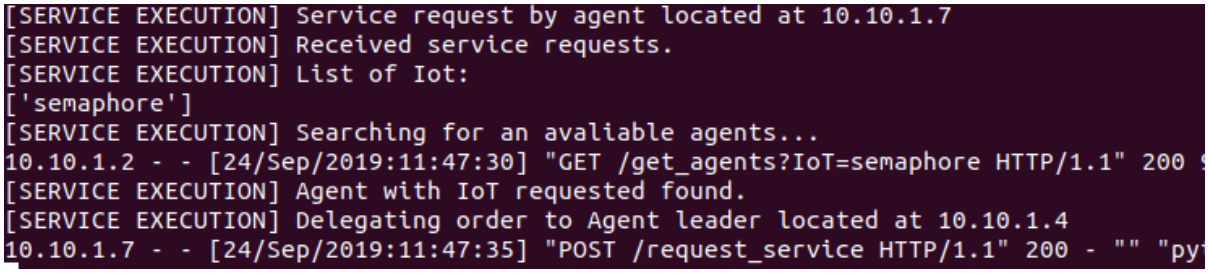


Fig. 53: Resolució de servei per l'agent cloud.

El segon leader realitza el procediment habitual i busca els agents que puguin executar el servei. En quest cas com solament un rep l'ordre d'execució degut a que l'escenari inicia. L'agent de l'altra zona rep l'ordre d'execució i es disposa a executar el servei. Com que aquest era l'únic capaç d'executar-lo informa al client dels resultats

7.6.4 Testeig notificació de resultats

Quan finalitza l'execució de cada servei es notifica al leader i al client. En la pantalla de l'aplicació del client es mostra el codi del resultat (èxit o fracàs), el nom del servei demanat i una descripció del que ha succeït.

En la pantalla del leader s'observa com aquest actualitza la disponibilitat del agent.

8. INTEGRACIÓ AMB ALTRES PROJECTES

Aquest projecte ha estat realitzat paral·lelament amb altres dos projectes relacionats també amb les Smartcities. El primer és "Conducción i calibración de dispositivos en un entorno Smarcity/F2C" realitzat per l'Adrian Rivas i l'Ilyas el Idrissi i el segon és "Sistema d'aparcament intel·ligent basat en F2C (Fog to Cloud)" realitzat per Nil Salvat.

Els tres projectes han sigut realitzats basant-se en el model Fog to Cloud d'una Smartcity. Els tres projectes requerien d'una plataforma basada en el model Fog to Cloud per poder executar serveis de la millor forma possible.

La integració realitzada en aquest projecte es basa en

- El funcionament de la plataforma formada per agents desenvolupats per altres estudiants.
- L'execució de serveis desenvolupats per altres grups.

8.1 Punts clau de la integració

Per tal de dur a terme una integració amb altres projectes s'han tingut que prendre unes decisions a nivell de disseny de la plataforma i dels agents.

8.1.1 API

L'API utilitzada en aquest projecte va ser desenvolupada per permetre la comunicació entre nodes els quals no compten amb canals de comunicació establerts. El fet d'utilitzar una API amb els mètodes relacionats amb els serveis amb les mateixes entrades i sortides és essencial per aconseguir una integració amb altres projectes.

8.1.2 Agregament d'agents

Per tal d'enregistrar agents implementats de manera diferent s'han realitzat modificacions al disseny de l'aplicació per tal d'**enregistrar agents mitjançant l'API**. Cal esmentar que els agents d'aquests altres projectes compten amb IoTs sense informació bàsica(que no es poden desar en les bases locals dels leaders) o

que no en compten amb cap. Per aconseguir això s'ha adaptat el mètode de register_agents de l'API.

```
if cherrypy.request.method == "POST":
    if self.agent.get_role() == "leader":
        body = cherrypy.request.json

        if 'IoT' in body:
            id_agent=  action.add_agent("",body['myIP'],self.agent.get_ip(),body['port'],"agent",body['ram'],body['cpu'],body['IoT'])
        else:
            id_agent=  action.add_agent("",body['myIP'],self.agent.get_ip(),body['port'],"agent",body['ram'],body['cpu'])
        return id_agent
```

Fig. 54: Mòdul API. Enregistrament d'agents.

8.1.3 Serveis

Els principals punts que s'han tingut en compte en el disseny de l'aplicació per tal d'executar serveis d'altres projectes són:

- **Definició de servei**: per poder executar serveis altres projectes s'ha utilitzat la mateixa definició bàsica d'un servei entre projectes. Aquesta definició ha sigut la mateixa per als **camps** que es troben sempre presents (dependències, codi principal, IoT...). A aquesta definició bàsica se li afegeixen els camps propis de cada servei.
- **Comunicació entre agents**: per aconseguir que agents externs executin serveis s'ha utilitzat l'API per tal realitzar peticions entre nodes.
- Les **entrades i sortides** de qualsevol execució han de ser les mateixes en tots els serveis. L'execució d'un servei es realitza amb la lectura dels paràmetres adients i en cas de que aquest retorni un resultat s'ha d'obtenir amb el format utilitzat en aquest projecte.

Per testejar aquesta integració s'ha portat a terme l'execució d'un servei implementat pel grup de conducció autònoma, el servei de calibratge de la il·luminació del test-bed.

La plataforma durant l'execució d'aquest servei tenia la següent forma:

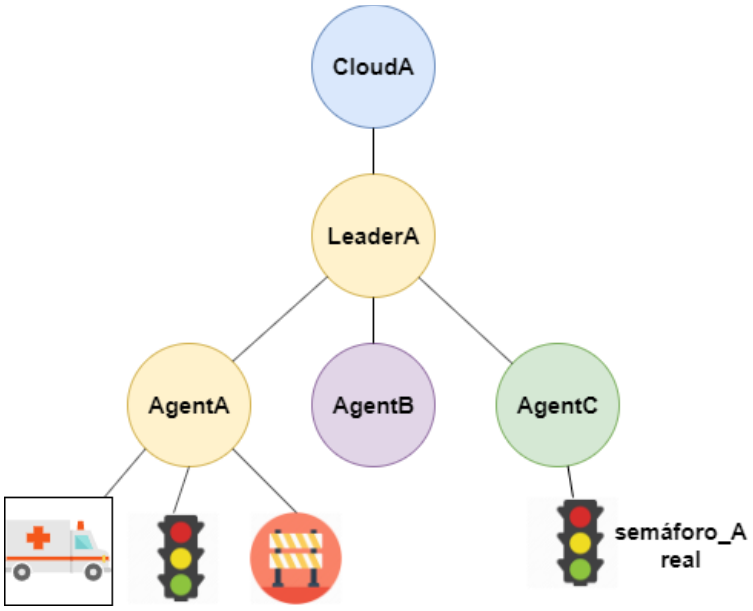


Fig. 55: Plataforma fog to cloud en la integració amb altres projectes.

La lletra A fa referència als agents implementats en aquest projecte, l'agentB és l'agent implementat per Nil Salvat i l'agentC pel grup de l'adrián Rivas i l'Ilyas el Idrissi.

El procés que s'ha portat a terme per executar aquest servei és:

1. Agregar el servei "CALIBRACION_SEMAFOROS_A" en el catàleg de serveis.
2. Enregistrar l'agentB (aquest pot ser utilitzat per serveis de computació) i l'agentC al leader de la plataforma a través de l'API.
3. L'usuari realitza la petició des d'un aplicació client (per realitzar aquesta petició s'ha utilitzat un navegado web).
4. La petició passa a ser atesa pel leaderA i dóna la ordre al agentC indicat en la definició del servei. Si aquest agent no es trobés disponible es notificaria del error.
5. Es dóna l'ordre d'execució al agent que compta amb el semaforo_A com a IoT.
6. Aquest agent executa el servei i notifica al client del resultat de l'execució.
7. El calibratge del il·luminat es visualitza: la llum d'aquestes parpelleja durant uns segons en un ordre determinat.

Aquest servei també pot ser executat com un servei simple ja que els agentsC s'agreguen amb els dispositius "semaforos_A".

9. DEVICE

En aquest apartat es descriu el procediment que s'ha seguit per dissenyar i construir un **device** o dispositiu (no confondre amb IoT) que pugui executar el software d'un agent.

9.1 Tecnologies usades

El disseny dels prototips 3D realitzats en els següents subapartats s'han realitzat amb l'aplicació web **Tinkercad**. Aquesta aplicació permet realitzar dissenys gràfics en tres dimensions basats en formes simples: cubs, circumferències, cilindres...

9.2 Raspberry pi

Un agent pot ser qualsevol node de la ciutat que compti amb un mínim de poder computacional i connectivitat de xarxa. A nivell de sistema qualsevol node que vulgui participar en la topologia ha ser capaç d'executar el software d'un agent.

Els requeriments esmentats son fàcilment acomplits en una ciutat intel·ligent on un agent pot ser un cotxe autònom, un ordinador portàtil o un telèfon mòbil. En aquest projecte s'ha utilitzat una **raspberry pi** com a exemple del que podria ser un dels agents de la topologia de la ciutat.

Una raspberry pi amb el software preinstal·lat necessari podria actuar com un agent de la plataforma administradora d'una Smartcity. Encara que aquest dispositiu compta amb algunes mancances a tenir en compte:

- **Autonomia**: pel funcionament d'una raspberry pi aquesta ha d'estar connectada a la corrent contínuament.
- **Connectivitat de xarxa**: una raspberry pi és com un ordinador de sobretaula: poden connectar-se a una xarxa wifi o tenir connexió a xarxa mitjançant un cable ethernet.
- **Visualització**: la raspberry és una màquina que no compta amb una pantalla incorporada. Per tal de visualitzar el sistema s'hauria de connectar un

monitor mitjançant un cable HDMI. A més, per interactuar amb aquesta s'hauria de connectar teclat i ratolí.

La majoria de dispositius d'avui en dia es poden utilitzar en entorns de mobilitat. Un portàtil o un telèfon mòbil amb connectivitat a xarxes wifi o 4G i amb una bateria incorporada poden utilitzar-se en qualsevol entorn. De manera similar en aquest projecte s'ha intentat resoldre les principals mancances de la raspberry .

En els següents subapartats es comenta com es solucionen aquests mancances i com s'ha dissenyat el Device a partir de les solucions aportades a aquests problemes.

9.3 Mòduls

La raspberry pi és un dispositiu que compta amb un ecosistema d'accessoris que ens permeten solucionar algunes de les carències esmentades anteriorment.
S'ha anomenat **mòdul** a tot aquell accessori de la pròpia Raspberry o complement de l'electrònica en general que permet solucionar alguna de les carències abans esmentades.

9.3.1 Powerbank

El primer mòdul a incorporar ens permetrà proporcionar **autonomia** energètica a la nostra raspberry quan aquesta no estigui connectada a la corrent. Una powerbank qualsevol ens permet proporcionar autonomia a un dispositiu mitjançant la connexió USB. Aquesta font d'energia ha d'estar carregada abans de connectar-la a la raspberry pi.

En aquest projecte s'ha utilitzat una powerbank amb les mesures més reduïdes possibles per tal de no perjudicar el disseny. Com que una raspberry consumeix poca energia qualsevol amperatge de powerbank és vàlid. Concretament s'ha utilitzat una powerbank de 2200 maH. Les mesures d'aquesta powerbank són 8,5 x 2,5 x 2,5 cm amb un pes de 84gr.

9.3.2 Power Over Ethernet

Un dels mòduls considerats a utilitzar ha sigut l'accessori propi de Raspberry, el Power Over Ethernet. Aquest mòdul permet el funcionament de la raspberry quan aquesta es troba amb el powerbank buit i no es disposa d'un carregador amb

connexió USB per a connectar la raspberry a la corrent.

Quan el mòdul PoE es troba acoblat amb la raspberry a aquest proporciona l'energia mitjançant la connexió d'aquest amb un cable ethernet d'un router operatiu. Com a contrapartida aquest mòdul requereix que el Device estigui fixe en un lloc per establir la connexió amb l'ethernet.

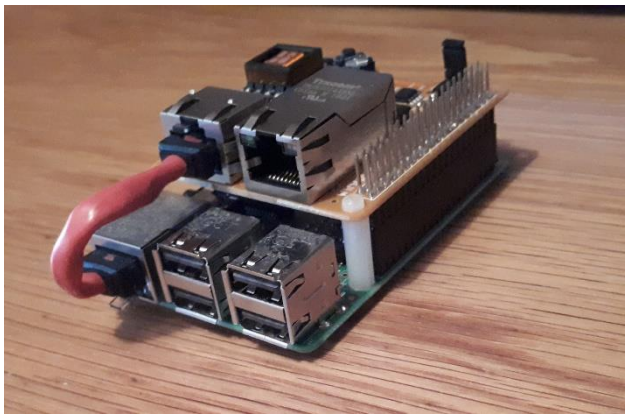


Fig. 56: Mòdul Power Over Ethernet acoplat a una raspberry pi.

9.3.3 Targeta SIM

El segon mòdul considerat a incorporar serà el que ens permeti tenir connectivitat 4G per l'ús del Device fora de xarxes wifi. Aquest mòdul disposarà entre d'altres components, d'un adaptador per a **targetes SIM** per realitzar connexions amb xarxes mòbils 4G.

La inclusió d'aquest mòdul permet la connectivitat a xarxa mitjançant una targeta SIM que compti amb gigabytes de dades.

Fig. 57: Mòdul targeta SIM: part frontal i posterior.

En aquest projecte no s'ha pogut comptar amb aquest mòdul degut a la poca distribució d'aquest. El disseny de prototip 3D dissenyat ni el construït finalment compten amb aquest mòdul.

9.3.4 Pantalla

Un dels altres mòduls és una pantalla tàctil que permet visualitzar i interactuar amb el software de la raspberry . Per aprofitar les funcionalitats d'aquesta pantalla s'hauria de comptar amb la versió d'escriptori de Raspbian.
Com que en aquest projecte el software del agent per a una raspberry s'executa directament al posar-se en marxa no s'ha vist necessari incloure aquest mòdul.

9.4 Disseny Device 3D

El fet d'incorporar aquests mòduls a una raspberry condiciona les capacitats i l'aspecte exterior d'aquesta. Per tal de garantir que els diferents components (raspberry i mòduls) no es facin malbé en l'execució d'un servei (per caigudes, contacte amb líquids, etc...) caldrà realitzar un disseny 3D d'un **recipient** o caixa que contingui tots els components escollits.

Aquest disseny 3D compta amb els següents requeriments:

- Respectar tant com sigui possible les **connexions** de la raspberry pi: tota connexió del propi hardware: com les connexió USB, jack 3.5mm i HDMI...
- Assegurar un cert nivell de **protecció** del dispositiu. Per si succeeix una caiguda sigui la "caixa" la que es faci malbé i no el hardware principal.
- **L'ocupació mínima** d'espai. Els agents estan pensat per ser utilitzats en entorns de mobilitat i la raspberry pi és un ordinador de mesures molt reduïdes. El disseny resultant haurà de contemplar que l'espai que ocupi el recipient sigui el mínim possible.

Com s'ha comentat anteriorment és necessari dissenyar un recipient on col·locar la raspberry i els mòduls utilitzats. La raspberry pi amb el mòdul PoE incorporat té una forma molt rectangular com es pot veure en la imatge 8.1 .

Aquests requeriments determinen que el disseny més apropiat sigui una "caixa" ajustada al hardware principal i a la powerbank. Aquesta caixa hauria de ser dissenyada per comptar amb forats amb les formes de les connexions allà on es trobin.

Per aconseguir això s'ha realitzat el disseny 3D d'un prototip. Aquest prototip està format per dues parts que s'acoblen una amb l'altra. La part inferior és on caldria acoblar la raspberry per a que aquesta es trobés fixe. Com que la part inferior de la raspberry es troben pins i les terminacions de connexions del propi hardware aquesta no s'hauria de trobar en contacte amb la superfície superior de la part inferior de la caixa. Per tal d'evitar aquest contacte s'hauria de comptar quatre punts que farien de plataforma on la raspberry es podria acoblar amb cargols.

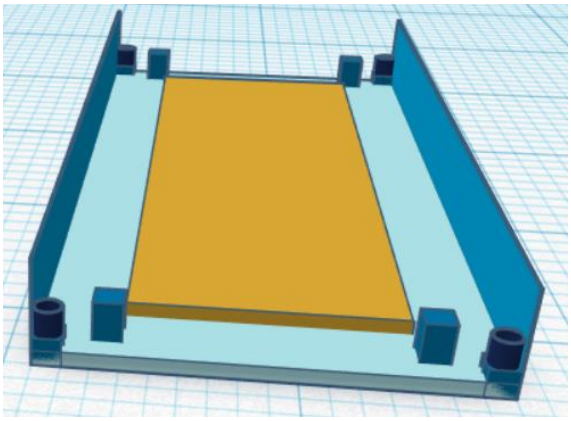


Fig. 58: Disseny prototip 3D part inferior. (La part taronja simula la posició de la bateria).

La part inferior a més compta amb uns petits forats als costats per on la part superior del recipient s'encaixarà. La part superior del recipient està dissenyada per acoblar-se amb la de sota però respectant les connexions de la raspberry. Durant el procés de disseny sempre s'ha tingut molt en compte les mesures i les posicions de les connexions de la raspberry pi.

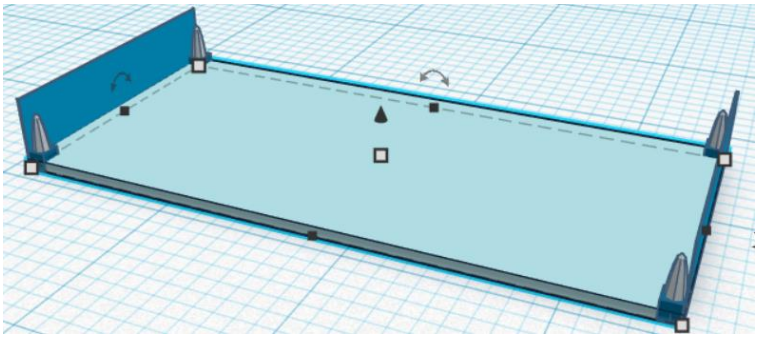


Fig. 59: Disseny prototip 3D part superior.

El disseny d'aquestes dues parts acoblades aconsegueix deixar un espai rectangular per a les connexions. De manera que en aquest disseny les connexions HDMI i jack

3.5 mm són accessibles però no les connexions USB. L'aplicació utilitzada Tinkercad no permetia dibuixar els forats amb la forma de les connexions per a què aquestes fossin més accessibles.

9.5 Construcció del Device

Com que durant el transcurs d'aquest projecte no s'ha comptat amb una impressora 3D per la impressió del prototip la construcció ha estat realitzada amb altres materials. La caixa o recipient construïda finalment s'ha fet amb fusta per un fuster professional.

Aquest disseny s'ha realitzat amb la raspberry pi i els mòduls PoE i powerbank. Com en el disseny anterior el hardware principal es troba acoblat de la mateixa manera: el mòdul Power Over Ethernet sobre la raspberry.

Per tal de protegir el hardware principal aquest s'ha col·locat a sobre de dues plataformes de fusta que permeten que no es produeixi contacte entre els pins de la part inferior de la raspberry i la part inferior de la capsa. Aquest contacte faria malbé tant la targeta SD com els pins de connexió.

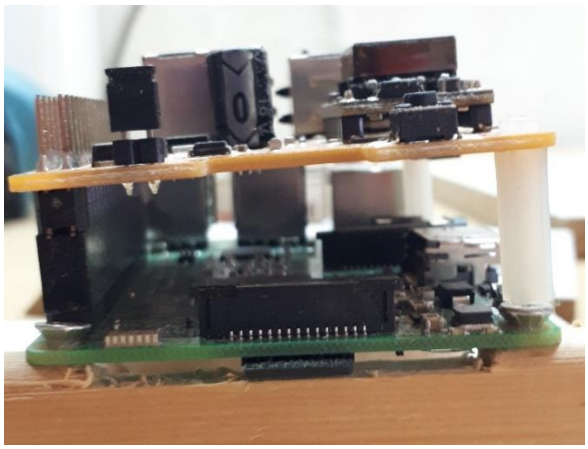


Fig. 60: Visualització de la part frontal del hardware principal del device.

La raspberry es troba connectada mitjançant un cable USB a la powerbank com a font d'alimentació trobant-se a l'esquerra del hardware principal, el més a prop possible a la connexió USB. Aquesta es troba subjectada en un espai rectangular de fet a mesura que permet una fàcil extracció i col·locació de la powerbank. Aquest

espai es troba elevat de tal manera que la powerbank quedi sobre la mateixa altura que el hardware principal. Com que la powerbank es pot treure o deixar per a carregar-se aquesta ha de ser fàcilment accessible i manipulable per l'usuari.

Fig. 61: Part interior del Device.

Les quatre connexions USB frontals del Device es troben disponibles per a la connexió de qualsevol altre component. Per veure o accedir a tots els components el Device compte amb un sistema d'obertura corredissa. Aquest es troba en la part superior del dispositiu i es basa en una peça lliscant.

Fig. 62: Part exterior del Device on es pot accedir a les connexions USB.

Aquest disseny acompleix part dels objectius proposats: soluciona el problema de l'autonomia i aquest es pot utilitzar en xarxes wifi.

Els punts forts del Device construït són:

- Es pot utilitzar estant el Device fixe en un espai o en un entorn de mobilitat on la **xarxa wifi** té un gran abast pels mòduls incorporats.
- L'usuari pot utilitzar els **USB** per connectar altres accessoris o dispositius.
- La **powerbank és fàcilment accessible** i es pot carregar fàcilment Encara que l'usuari hauria de retirar la part superior de la capsa.
- Degut a la pròpia construcció el Device és sòlid i robust i l'usuari no s'ha de preocupar perquè es faci malbé si el dispositiu es cau.
- EL Device es pot carregar al mateix temps que s'està utilitzant.

Inconvenients d'aquest disseny:

- Com a contrapartida de la robustesa tenim un dispositiu que podria tenir unes **mesures** molt més reduïdes. En el següent apartat es comenta el principal problema que ha impedit obtenir un disseny òptim.
- Les connexions jack 3.5 mm i HDMI no es poden utilitzar degut a la col·locació del powerbank.
- S'ha de retirar la tapa per a carregar el dispositiu.

No s'ha considerat ni com a punt fort ni com a desavantatge la extracció i col·locació de la targeta SD. Aquesta pot ser extreta amb dificultats però la col·locació solament pot ser realitzada si la persona sap on està l'espai de col·locació de la targeta de la part inferior de la raspberry. Un usuari final no tindria accés a aquesta targeta.

9.6 Comentaris sobre el disseny i construcció

El disseny i construcció d'aquest Device ha sigut realitzat per persones que no són professionals en el disseny de dispositius electrònics.

El que s'ha intentat aconseguir amb aquest Device o dispositiu és exemplificar que qualsevol màquina ja sigui telèfon mòbil, raspberry pi o ordinador podria ser capaç de ser un agent sempre i quan pogués córrer l'aplicació desenvolupada.

Encara i no ser un expert en el disseny de dispositius considero que aquest **disseny podria ser millorat** si és comptés amb els recursos necessaris. Si es comptés amb una bateria com la que porten els telèfons mòbils directament connectada a la

raspberry aquesta podria ser col·locada en la part inferior de la plataforma. De manera que la bateria quedaria entre la part inferior de la raspberry i la part inferior de la caixa en si.

Això reduiria en gran mesura les dimensions del Device construït però solament pot ser realitzat per algun professional d'electrònica com s'ha comentat anteriorment. Aquesta bateria incorporada podria ser com la de qualsevol dispositiu electrònic que es port carregar directament fora de la caixa mitjançant una connexió micro-USB o USB tipus C.

Una altra millora de disseny que no s'ha pogut desenvolupar finalment és la incorporació de la càrrega de la powerbank sense retirar la tapa lliscant. Aquesta càrrega es portaria a terme mitjançant un adaptador micro USB mascle i femella. Seria necessari per això modificar la capsa per comptar amb un forat on l'usuari podria carregar el Device.

9.7 Interacció amb el Device

Com s'ha comentat anteriorment el Device està configurat per executar el software d'un agent de manera automàtica. Junt amb l'execució de cada agent cada node disposa d'una aplicació client on cada client pot sol·licitar els serveis que vol executar. Amb el Device no es pot interactuar de la mateixa forma que amb un ordinador o un telèfon mòbil.

En aquest projecte s'ha realitzat una petita aplicació exactament igual que la aplicació client de cada agent. Aquesta aplicació s'ha d'executar en una màquina diferent a la raspberry que compti amb pantalla i teclat com qualsevol ordinador.

Com que cada agent al executar-se agafa una adreça IP amb la que executar-se automàticament seria possible **realitzar peticions de servei a aquest agent si aquesta adreça** fos coneguda. Per saber la direcció IP d'un dispositiu en una xarxa seria suficient en comprovar l'assignació d'adreces realitzades en aquella xarxa.

Es pot comprovar de diverses maneres però en aquest projecte s'ha utilitzat l'aplicació mòbil Fing per trobar la direcció IP de la raspberry, com es veu a continuació:

Fig. 63: Aplicació mòbil fing.

Una vegada des de l'aplicació client es coneix l'adreça ip que està utilitzant el Device seria possible realitzar peticions de servei a través de l'aplicació client.

10. TREBALLS FUTURS

Aquest projecte s'ha centrat en el desenvolupament de la capa Fog de la topologia d'una Smartcity. Com a proposta de continuació d'aquest projecte es podria realitzar un desenvolupament d'una capa Cloud formada per diversos nodes. De manera similar com s'ha realitzat en la capa fog per efectuar més àgilment les tasques d'aquesta capa.
De la mateixa forma que la capa Cloud la capa de dispositius s'ha desenvolupat mínimament. Com a millora del projecte i la realització de proves en entorns on els dispositius es compten per milers es podria realitzar una simulació de la capa IoT amb alguna eina de virtualització com Docker. La virtualització de dispositius permetria establir protocols de comunicació entre agents i IoT.

Els serveis realitzats en aquest projecte han sigut desenvolupats per testejar la topologia presentada. Partint de les resolucions de serveis que s'ha dut a terme en aquest projecte poden aparèixer nous tipus de serveis que requereixin resolucions o execucions diferents.

L'aplicació client que s'ha utilitzat tant pels agents com per realitzar peticions a la raspberry que actua com agent és suficient per realitzar el testeig del projecte. Ara bé, seria millor de cara a un usuari final comptar amb una aplicació gràfica que permetés la petició de serveis més còmodament.

El front-end desenvolupat permet l'acompliment bàsic dels requeriments del projecte. Encara que podria ser millorat mitjançant una millor visualització de la informació topològica de la plataforma. Addicionalment, es podria incloure alguna funcionalitat més com la modificació i l'agregació de serveis.

Un altre punt molt important a tenir en compte és la seguretat de la topologia, en aquest projecte no s'ha realitzat una política de seguretat. Per finalitzar el projecte seria indispensable desenvolupar aquest punt seguint una política de seguretat que s'adapti al disseny de plataforma realitzat.

11. CONCLUSIONS

Considero que els objectius plantejats inicialment han sigut complerts satisfactòriament degut a que aquests s'han dissenyat en múltiples iteracions del projecte. Cadascun d'aquests objectius ha sigut assolit perquè contínuament s'ha analitzat, modificat i comprovat que cadascun d'ells funcionava d'acord a un disseny lògic i estudiat.

Considero que la plataforma dissenyada acompleix els requisits inicialment capturats d'aquesta. Aquesta plataforma s'ha dissenyat i construït en un escenari de màquines lògic i estudiat. Encara i no ser un expert en el disseny de plataformes basades en grans volums d'informació considero que l'escenari resultant és funcional d'acord una de les finalitats marcades en aquest projecte: l'execució de serveis de la millor manera possible.

L'objectiu al que se l'hi ha donat més pes al llarg del projecte és a l'aplicació dels agents. Aquesta aplicació s'ha anat modificant durant el transcurs del projecte per tal de que aquesta garanteixi el funcionament de la plataforma a nivell tècnic el millor possible. Considero que el producte final encaixa amb el que s'havia pensat en un inici i crec que el software dissenyat podria ser utilitzat per altres desenvolupadors amb facilitat.

Considero que el fet de treballar paral·lelament amb altres projectes del mateix àmbit ha sigut clau per tal d'aconseguir un producte més rodó. El fet de testejar el projecte desenvolupat conjuntament amb altres persones sempre serveix per millorar i acabar de polir el disseny d'un producte.

D'aquests objectius la construcció i disseny del Device ha sigut un dels que el resultat final ha variat de com es plantejava en un inici. Degut a la difícil obtenció dels mòduls i de les limitacions en quant a recursos (la impressora 3D) s'ha arribat a un resultat diferent al que s'esperava. Tot i això considero que el resultat és positiu i que aquestes dificultats m'han ensenyat a buscar solucions alternatives. Considero que amb el disseny i construcció d'aquesta màquina s'ha aconseguit l'objectiu d'aquesta:

demostrar que una màquina qualsevol amb el software adient pot ser un node capaç d'administrar una ciutat intel·ligent.

11.1 Valoració personal

Considero que aquest projecte ha estat realitzat satisfactòriament però considero que la organització ha sigut un dels punts a millorar. La planificació en molts casos ha sigut molt optimista i en ocasions no s'ha arribat a completar tots els punts plantejats inicialment en un sprint.

Crec que s'hauria d'haver invertit més temps en realitzar una planificació més realista segons el temps amb el que es comptava en un sprint. D'aquesta manera els tutors haurien seguit millor els avenços aconseguits al llarg del projecte.

En general considero que hi ha hagut un aprenentatge professional i personal amb aquest projecte. El fet de seguir una metodologia de treball m'ha ajudat personalment a organitzar-me i a planificar. Encara que considero que encara em falta pràctica en aquests aspectes.

12. PRESSUPOST DEL PROJECTE

En aquest apartat es detalla quin és el cost total del projecte si aquest es posés a la venta al mercat. En aquest cas es posaria a la venta: la plataforma en si, el software d'un agent per a que el client pogués executar els serveis que volgués.

Per calcular el cost del projecte s'ha definit un preu per cada hora dedicada. He considerat que un preu just seria 15 € l'hora. Si tenim en compte que aquest projecte s'ha desenvolupat en 560 hores:
PreuTotal = 560 h * 15€/h = **8400€.**

Si el client desitgés que el manteniment del servei Apache2 i per tant del front-end es trobés al meu càrrec se l'hi hauria d'afegir un preu mensual pel consum energètic i de recursos . Calculo que aquest preu seria d'uns **200€** per mes.

13. BIBLIOGRAFIA

CherryPy A minimalist Python Web Framework. Disponible en:
https://cherrypy.org/

Socket, a Low-level networking interface. Disponible en
https://docs.python.org/3/library/socket.html

MongoDB driver. Disponible en:
https://www.php.net/manual/en/set.mongodb.php

MySQLdb User's Guide. Disponible en:
https://mysqlclient.readthedocs.io/user_guide.html

Python MongoDB. Disponible en:
https://www.w3schools.com/python/python_mongodb_getstarted.asp

Subprocess – Subprocesses management
https://docs.python.org/3/library/subprocess.html

Threading- Thread-based parallelism. Disponible en:
https://docs.python.org/3/library/threading.html

14. REFERÈNCIES

[1] Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions). Disponible en: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ [consulta octubre 2019]

[2] Wikipedia, Smart city . Disponible en: https://en.wikipedia.org/wiki/Smart_city [Consulta Desembre 2018]

[3] Smartcities face challenges and opportunities. Disponible en: https://www.computerweekly.com/opinion/Smart-cities-face-challenges-and-opportunities [Consulta Desembre 2018]

[4] Smart City Architecture and its Applications Based on IoT. Disponible en: https://www.sciencedirect.com/science/article/pii/S1877050915009229 [Consulta Desembre 2018]

[5] The Role of Internet of Things (IoT) in Smart Cities: Technology Roadmap-oriented Approaches. Disponible en: https://www.researchgate.net/publication/324915158_The_Role_of_Internet_of_Things_IoT_in_Smart_Cities_Technology_Roadmap-oriented_Approaches [Consulta Desembre 2018]

[6] The framework of smart city based on cloud computing https://www.tsijournals.com/articles/the-framework-of-smart-city-based-on-cloud-computing.pdf [Consulta Desembre 2018]

[7] The Role of Cloud of Things in Smart Cities https://www.researchgate.net/publication/311911547_The_Role_of_Cloud_of_Things_in_Smart_Cities [Consulta Desembre 2018]

[8] Smart cities and cloud computing: lessons from STORM CLOUDS experiment https://www.researchgate.net/publication/302978557_Smart_Cities_and_Cloud_C

omputing Lessons from the STORM CLOUDS experiment [Consulta Desembre 2018]

[9] Towards an Open, Secure, Decentralized and Coordinated Fog-to-Cloud Management Ecosystem (mF2C). Disponible en: https://www.mf2c-project.eu/wp-content/uploads/2018/02/Whitepaper_F2C_v2.pdf [Consulta Desembre 2018]

[10] MQTT version 3.1.1 client class. Disponible en: https://pypi.org/project/paho-mqtt/#description [Consulta febrer2019]

[11] Python P2P networking library, pyp2. Disponible en: https://pypi.org/project/pyp2p/ [Consulta febrer2019]

[12] What is Twisted? Disponible en: https://twistedmatrix.com/trac/ [Consulta febrer 2019]

[13] Socket, a Low-level networking interface. Disponible en https://docs.python.org/3/library/socket.html [Consulta febrer2019]

15. ANNEXOS

Trobarem dos annexos principals en aquest projecte:

- El primer annex tracta sobre la investigació i justificació d'algunes de la llibreria utilitzada en aquest projecte per a la implementació de la comunicació entre nodes.
- El segon és un manual tècnic on s'expliquen els passos a seguir per poder instal·lar i configurar el software necessari utilitzat en aquest projecte per tal de reproduir la plataforma dissenyada.

Esmentar que tot el software utilitzat en aquest projecte s'ha instal·lat i configurat en una màquina virtual Linux (versió 18.04) a excepció de les configuracions realitzades en la raspberry pi.

15.1 Estudi de llibreries en python3

Una de les decisions principals que es van prendre en el projecte es quin protocol o llibreries s'utilitzaven per a la comunicació entre agents. Encara i que la API s'utilitza per a la petició i resposta de serveis no permet l'establiment de canals de comunicació entre màquines diferents.

A continuació es defineixen breument les llibreries que es van considerar utilitzar en aquest projecte per tal de comunicar els agents entre si, així com la justificació de la llibreria escollida.

15.1.1 Mosquitto

Mosquitto[10] permet la comunicació entre nodes mitjançant l'intercanvi de missatges entre ells. Aquesta llibreria està basada en el **protocol MQTT,** aquest és un protocol de connectivitat màquina-a-màquina dissenyat per efectuar un intercanvi de missatges mínim basat en un mètode de publish/subscribe de transport.

Aquest protocol i llibreria estan pensades per ser utilitzades quan la màquina en qüestió té poc poder computacional (una raspberry, un ordinador antiquat...) i la connectivitat a xarxa és insuficient.
Addicionalment, en la pàgina de suport de Mosquitto de llibreries de python s'ofereixen exemples d'ús que permeten la comunicació entre màquines clients i servidor.

15.1.2 PyP2P

PyP2P[11] és una llibreria de xarxa pensada per a construir xarxes **peer to peer** en python. El principal punt fort d'aquesta llibreria és que s'encarrega d'acceptar les connexions i d'eliminar les antigues automàticament. Entre altres coses permet la definició de respostes automàtiques mitjançant missatges. Addicionalment, aquesta llibreria ofereix facilitats per trobar nodes que es troben darrere d'un NAT.
Aquesta llibreria està basada en **el protocol TCP.**

Com en el cas anterior, en la pàgina oficial de suport d'aquesta llibreria s'ofereixen algun exemple d'implementació molt simple. Actualment aquesta llibreria es troba en fase de proves i des de la pàgina oficial s'informa que pot presentar bugs.

15.1.3 Twisted

Twisted[12] és una llibreria de xarxa basada en la **resposta d'events**. Els mètodes de connexió, desconnexió i notificació de missatges són definits com events de l'aplicació. Aquesta llibreria també permet el mètode d'intercanvi de missatges publish/subscribe.

La pàgina oficial de twisted ofereix diversos exemples de diferents implementacions basades en el mètode publish/subscribe i en la resposta d'events. Aquesta llibreria suporta els protocols UDP, TCP i SSL.

Aquesta llibreria està pensada per ser utilitzada junt amb altres protocols, per tal de crear una integració entre protocols . És a dir que permet la comunicació amb màquines que utilitzen protocols diferents (o això s'indica des de la seva pàgina oficial).

Aquesta llibreria actualment es troba en continu desenvolupament (l'última versió està disponible des d'abril de 2019).

15.1.4 Socket

Socket[13] és la llibreria bàsica de comunicació entre màquines que ofereix Linux per defecte. Es basa en establir una comunicació seguint el model **client i servidor**. Els protocols que permet són TCP i UDP.

El principal punt fort d'aquesta llibreria és que és **molt usada** i la documentació publicada és extensa. Qualsevol error ocorregut en la implementació pot ser fàcilment resolt degut a aquest suport.

15.1.5 Comparació i justificació d'ús

Com es pot veure en la taula següent, es fa un resum de les llibreries considerades per implementar la comunicació entre màquines:

LLibreria	Protocols suportats	Característiques principals	Comentaris addicionals
Mosquitto	MQTT, sobre TCP	- Basat en el mètode publish/subscribe. - Efectua un intercanvi de missatges en temps real.	- Suporta l'encriptació.
Pyp2p	TCP	- Accepta i elimina connexions automàticament segons l'estat del canal de comunicació.	- Es troba en fase experimental.
Twisted	TCP, UDP, SSL	- Es basa en la resposta d'events.	- Llibreria actualitzada. - També permet publish/subscribe
Socket	TCP i UDP	- Basada en el model client servidor.	- Documentació de llibreria extensa.

Taula 7: Resum de llibreries de python3 per a l'establiment de les comunicacions.

Mosquitto és una llibreria pensada per a comunicar màquines poc potents on hi ha limitacions de xarxa. Si es desitgés construir la plataforma solament amb dispositius de pocs recursos com raspberrys aquesta seria ideal. Però en una ciutat intel·ligent es poden trobar tot tipus de màquines amb més o menys recursos.

Pyp2p és una llibreria que es troba en fase experimental i per un projecte d'aquesta magnitud no seria la millor opció. Com que aquesta llibreria per defecte ja té implementats alguns procediments (la detecció de caigudes) caldria comprovar si aquests poden ser modificats.

Twisted és una llibreria que compta amb un fort desenvolupament . A més des de la seva pàgina oficial hi ha implementacions exemplars de connexions TCP i UDP. Com que es basa en la resposta d'events aquesta estalviaria al desenvolupador la implementació d'alguns mètodes o procediments a seguir. El principal problema que té és que no compta amb tant suport en fòrums pels errors d'implementació que poden succeir.

En aquest projecte s'ha decidit utilitzar la llibreria **socket** per a la comunicació entre agents. Aquesta permet fer possible establir les comunicacions desitjades entre els nodes de la topologia comentades en l'apartat Comunicació entre nodes.
La principal raó d'elecció és que aquesta llibreria és la que compta amb més **documentació** en pàgines oficials i en fòrums pels errors d'implementació. Aquesta avantatja ha permès al desenvolupador del projecte cercar fàcilment si les comunicacions necessàries de la plataforma es podien implementar o no.

15.2 Instal·lació i configuració d'un agent

En aquest apartat es descriuen tots els passos a seguir per poder executar correctament l'aplicació dels agents en un màquina Linux.

Tant en aquest apartat com en el següent la majoria de passos solament poden ser portats a terme per un usuari amb **permisos root** o amb la comanda "sudo" davant de la comanda en qüestió a realitzar. La comanda sudo ens permet executar com a administrador la comanda indicada després del sudo.

Com a recomanació i per verificar el funcionament de cadascun dels serveis es recomana efectuar les comprovacions que s'indiquen al final de cada explicació.

15.2.1 Instal·lació i configuració MySQL

Primerament començarem per la instal·lació del servei client i servidor de MySQL. MariaDB, una versió de MySQL.

$ apt-get install mariadb-server mariadb-client

És altament recomanat canviar la contrasenya del usuari root de MySQL:

$ mysql_secure_installation

Aquesta comanda també permet canviar altres paràmetres per defecte, com el del directori arrel del servidor. Seguidament accedirem al servei Mysql mitjançant l'usuari root per crear l'usuari que utilitzaran els agents:

$ mysql -u root -p

Primerament es sol·licitarà la contrasenya de la màquina i després la de MySQL indicada anteriorment.

Procedirem creant una base de dades i un usuari amb els permisos adients. Dins de mysql efectuarem:

CREATE DATABASE resources;
CREATE USER nom_usuari IDENTIFIED BY 'contrasenya'
GRANT ALL PRIVILEGES ON * TO 'nom_usuari'@'%' IDENTIFIED BY

'contrasenya' ;
FLUSH PRIVILEGES;
EXIT;

Per permetre la connexió externa de l'administrador modificarem els arxius de configuració del servidor. En aquest arxiu podem canviar diversos paràmetres per defecte del servidor.

$ nano /etc/mysql/mariadb.conf.d/50-server.cnf

bind-address 0.0.0.0

En el paràmetre bind_address indicarem l'adreça de la màquina (0.0.0.0) per tal de permetre connexions locals i externes. Per aplicar els canvis aplicats caldrà reiniciar els servei mysql:
$ service mysql restart

L'usuari i contrasenya indicats en aquests passos han de coincidir amb els que utilitza el mòdul **resources** per accedir a la base de dades indicada. A més, com que l'**administrador** també realitza connexions remotes des de el front end aquest també ha de tenir aquestes dades.

Per a comprovar que es pot accedir remotament a la base de dades d'un agent caldrà comptar amb una altre màquina que tingui el servei client de MySQL o que estigui configurada també com a agent.
En aquesta màquina externa realitzarem una connexió remota a aquesta base de dades creada de la mateixa manera que els agents. Per fer-ho mitjançant el terminal de comandes caldria fer:
$ mysql nom_usuari@host

15.2.2 Instal·lació i configuració MongoDB

De manera similar a MySQL procedirem a configurar el servei MongoDB de l'agent Cloud. Començarem amb la instal·lació del paquet base i servidor:
$ apt-get install mongodb mongodb-server

Com en MySQL modificarem el fitxer de configuració de MongoDB per permetre l'accés remot. Per això modificarem el fitxer de configuració del servidor:

$ nano /etc/mongodb.conf

bind-ip= 0.0.0.0

Un cop indicada l'adreça ip de ala màquina reiniciarem el servei de mongoDB per aplicar els canvis:

$ service mongodb restart

MongoDB no requereix d'un usuari per accedir a la base de dades utilitzada ni tampoc cal crear-ne una a diferència de MySQL.

Com en MySQL cal assegurar-se que l'adreça ip de l'agent cloud utilitzada sigui la mateix que la que té indicada els mòduls **app_agent** i **api** i el programa **init_services.py**. Degut a que els tres mòduls es connecten a les bases de dades de l'agent cloud.

Per comprovar que el servei mongoDB funciona correctament efectuarem la següent comanda:

$ service mongodb status

Si aquest servei es troba operatiu ens apareixerà per pantalla "active (running)" i significarà que la base de dades topològica i de serveis podran ser accessibles remotament.

15.2.3 Llibreries python3

El software dels agents utilitza llibreries externes per a efectuar algunes tasques. Per instal·lar aquestes llibreries es pot utilitzar l'instal·lador de paquets de python3 pip. Totes les llibreries esmentades a continuació es poden descarregar utilitzant la següent comanda:

$ pip3 install nom_llibreria

Les llibreries utilitzades en aquest projecte són: psutil, pymongo, cherrypy (API), mysqlclient i pysftp.

15.2.4 Enviament de dades

Per tal de que cada agent pugui enviar la seva informació correctament s'haurà de realitzar una connexió al servidor SFTP de l'administrador. Aquest servidor ha estar instal·lat i configurat en una màquina com s'explica en l'apartat 9.4 Configuració d'un servidor SFTP.

Un cop aquest servidor està configurat utilitzaríem la següent comanda per l'enviament de dades:

$ sftp -P port usuari@adreça_ip_admin

Si el port utilitzat pel servidor és el 22 no serà necessari indicar aquest paràmetre.

Aquest pas ha d'estar realitzat també per l'agent cloud a si mateix per tal de carregar els diversos fitxers dels serveis.

15.3 Raspberry pi

A continuació s'explica com posar en marxa una raspberry pi sense cap sistema operatiu instal·lat en la targeta SD. Un cop la raspberry és operativa i funcional es descriu com configurar-la per a que executi el software d'un agent al posar-se en marxa.

14.3.1 Instal·lació i configuració del sistema operatiu

Per tal de posar una raspberry en marxa cal disposar dels següents components:

- Raspberry Pi, amb el seu carregador.
- Tarjeta de memòria micro SD, almenys d'uns 4GB.
- Cable HDMI (primer cop).
- Teclat amb connexió USB (primer cop).
- Cable Ethernet.

Els dispositius de la família Raspberry disposen d'un sistema operatiu propi anomenat **Raspbian**, basat en Debian. En aquest apartat s'explica com instal·lar-lo en un dispositiu de la família Raspberry qualsevol, en cas de que no el tinguem instal·lat.

Raspbian és un sistema operatiu gratuït que podem descarregar en la pròpia pàgina oficial de Raspberry:
https://www.raspberrypi.org/downloads/raspbian/

Hi ha 3 versions disponibles: una amb escriptori i aplicacions preinstal·lades , una amb versió d'escriptori normal i una altra sense escriptori i per tant que funciona mitjançant el terminal de comandes. En condicions normals no es recomana la versió sense escriptori degut a que aquesta solament la poden utilitzar desenvolupadors correctament. Per a executar el software d'un agent aquesta versió és suficient amb la versió sense escriptori encara que les altres opcions són igualment vàlides. Comentar que les versions amb escriptori ocupen més espai de disc, fet a tenir en compte si la targeta SD té una capacitat de 4 GB.

Una vegada em escollit l'opció que més s'escau a les nostres necessitats passarem a la descàrrega d'aquesta, amb extensió zip. Descomprimirem la imatge de Raspbian en una carpeta que formi part d'un camí de fàcil localització de la nostra màquina (l'escriptori, documents...).

Procedirem col·locant la targeta microSD de la raspberry en un adaptador de targetes SD, en el lector de targetes de la nostra màquina. En cas de que tinguem algun arxiu en aquesta targeta la formatem.

Per a que la raspberry s'engegui amb el sistema operatiu Raspbian utilitzarem un instal·lador que ens escrigui la imatge del sistema operatiu en la targeta de memòria. Per exemple **balenaEtcher,** que podem trobar-lo en:
https://www.balena.io/etcher/
Quan aquest programa s'acabi d'instal·lar l'executarem i li indicarem on es troba la targeta microSD de la raspberry (el camí) i la imatge de Raspbian instal·lada anteriorment, fent així el *flashing* de la targeta.

Pot ocórrer que al arribar en aquest punt encara i haver realitzat els passos anteriors no se'ns visualitzi en la pantalla l'execució del sistema operatiu de la raspberry. Això és degut a que la raspberry no detecta la connexió HDMI i caldrà indicar-l'hi que ho

detecti. Per fer-ho cal modificar un fitxer de configuració del software instal·lat en l targeta SD anomenat config. Al final d'aquest fitxer li indicarem el següent:

$ nano config.txt (dins de la targeta SD)

hdmi_force_hotplug = 1

A continuació passarem a col·locar la targeta en la raspberry i passarem a la pròpia configuració del software.

Connectarem la raspberry amb un cable ethernet a un router operatiu, per tal de que se'ns assigni una adreça ip. Per visualitzar l'execució podem connectar la raspberry amb un monitor o televisor mitjançant un cable hdmi i mitjançant un usb 2.0 un teclat. Aquest procediment es realitza estant la raspberry connectada a la corrent mitjançant el seu carregador. Mencionar que la connexió ssh per defecte està deshabilitada per lo qual és recomanable primer usar teclat i pantalla, habilitar la connexió ssh i després treballar amb la raspberry des d'una màquina externa.

Seguidament s'explica com realitzar la **configuració inicial del sistema operatiu** aquesta pot ser realitzada més fàcilment si em instal·lat una versió d'escriptori.

Per pantalla ens demanarà un login, li indicarem l'usuari 'pi' de contrasenya 'raspberry' . Cada raspberry té aquest usuari i contrasenya per defecte. El que també s'aconsella és indicar una nova contrasenya a l'usuari root, degut a que d'aquí endavant usarem comandes que sol pot realitzar l'administrador. Seguidament procedirem actualitzant els paquets del sistema.

$ sudo passwd root

$ apt-get update

$ apt-get upgrade

Seguidament, es recomana canviar el llenguatge per defecte del teclat a l'espanyol i català degut a que els caràcters especials com la ç no apareixen. Per fer-ho modificarem el fitxer keyboard de configuració. EN el camp indicat a sota indicarem el llenguatge espanyol:

$ nano /etc/default/keyboard

XKBLAYOUT0="es"

Finalment reiniciarem el sistema per aplicar els canvis de teclat realitzats.

$ reboot

Procedirem amb la configuració del servei ssh per poder accedir a aquesta màquina externament. Per fer-ho serà necessari instal·lar el següent paquet:

$ apt-get install openssh-server

Procedirem a modificar el fitxer de configuració del servidor per permetre l'accés remot, serà suficient canviant el port per defecte:

$ nano /etc/ssh/sshd_config

Port 2200

Per defecte el servei de **ssh no s'executa al posar en marxa** la màquina. Per fer-ho cal executar la següent comanda:

$ update-rc.d ssh enable

Si reiniciem el servei ssh podrem procedir amb la instal·lació del codi:

$ service ssh restart

15.3.2 Instal·lació de codi

Per aconseguir que una raspberry s'executi com a agent aquesta ha de comptar amb el codi necessari. Des d'una màquina externa s'ha de pujar el software de l'aplicació. Per pujar el conjunt de fitxers de l'aplicació utilitzarem el servei ssh configurat anteriorment.

Accedirem remotament al servidor sftp de la raspberry mitjançant l'usuari pi i l'adreça ip de la raspberry:

$ sftp -P 2200 pi@adreça_ip_raspberry

Dins del servidor crearem un directori on s'instal·larà el codi i pujarem els fitxers:

$ mkdir /agent

$ cd agent

$ put *

L'asterisc * vol dir que es pujaran tots els arxius del directori on ens trobem de la màquina externa.

15.3.3 Execució com agent

Quan la raspberry té tot el software necessari caldrà configurar-la perquè s'executi com a agent al engegar-se. Per fer això cal especificar a nivell de sistema operatiu que el software d'agent s'executi al inici. Esmentar que la següent explicació solament ha funcionat amb el **sistema operatiu Raspbian**, no en Linux.

Utilitzarem el cron de la màquin per a la programació de l'execució cada vegada que aquesta s'engegui. El **cron** és la llista de tasques programades per un usaru que s'executen cada un període de temps indicat. Aquest pot ser accedit mitjançant la següent comanda:

$ crontab -e

@reboot python3 /path/app_agent.py –-r a &

La directiva @reboot permet que s'executi el software al posar-se en marxa la màquina. El paràmetre "--r" indica el rol ("a" d'agent, "l" de leader i "c" de cloud). El caràcter '&' permet que el software s'executi en **background**.

Per comprovar que l'execució és vàlida executarem la següent comanda una vegada el sistema operatiu s'ha posat en marxa:

$ ps aux | grep app_agent

Ens hauria d'aparèixer per pantalla el procés, així com uns valors de consum de ram i cpu superiors a 0.1. Si la màquina no comptava amb connectivitat a xarxa en un moment inicial l'execució com a agent no funcionarà.

15.4 Configuració del servidor web Apache

Per dissenyar i visualitzar un front-end cal comptar amb un servidor web que ens ho permeti. Hi ha diversos servidors web però en aquest projecte s'ha utilitzat l'Apache, un servidor web per defecte instal·lat en Linux i Debian.
En cas de que no es tingui instal·lat caldrà instal·lar el paquet de sistema corresponent:

$ apt-get install apache2

Començarem creant la carpeta arrel on es col·locaran tots els fitxers necessaris, amb els respectius permisos de lectura i escriptura per part l'usuari de la màquina:

$ mkdir -p /var/www/miau_admin.com/html
$ chown -R $USER:$USER /var/www/miau_admin/html
$ chmod -R 755 /var/www/miau_admin.com

Per visualitzar el front-end caldrà declarar i configurar un **host virtual** . Els hosts virtuals són el mètode que s'utilitza per allotjar múltiples pàgines web en un mateix servidor. Per fer-ho crearem un fitxer de configuració per aquest host virtual, com el que es veu en la figura 1.

$ nano /etc/apache2/sites-avaliable/miau_admin.com.conf

```
<VirtualHost *:80>

        ServerAdmin onlyrxdyz@admin.com
        ServerName miau_admin.com
        ServerAlias www.miau_admin.com
        DocumentRoot /var/www/miau_admin.com/html
        DirectoryIndex blank.php blank.php

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Fig. 64: Declaració d'un host virtual del servidor apache.

En aquest fitxer s'indiquen els paràmetres de la pàgina web per defecte com per exemple el nom de la pàgina web (ServerName) o el fitxer arrel (DocumentRoot) del front-end.

Després de guardar els canvis serà necessari habilitar el lloc web i reiniciar el servei per aplicar els canvis. També desactivarem el lloc web proporcionat per defecte per Apache:

$ a2dissite 000-default.conf

$ a2ensite miau_admin.com.conf

$ systemctl restart apache2

Procedirem amb indicar el nom del lloc web en un fitxer a part, per tal d'evitar possibles errors en la posada en marxa del servidor:

$ echo "Servername" miau_admin.com | sudo tee /etc/apache2/conf-avaliable/servername.conf

$ a2enconf servername

Per comprovar que tot ha anat bé accedirem a la pàgina web des d'un navegador web:

http://qualsevol adreça ip màquina:port

Si el port utilitzat en el host virtual és el 80 no serà necessari indicar-ho. Si per algun motiu el servidor no s'ha pujat en marxa podem executar la següent comanda per tenir més informació al respecte:

$ apache2ctl configtest

Per finalitzar la posada en marxa del front-end serà necessari instal·lar els paquets que utilitza aquest per a la monitorització d'agents:

$ apt-get install php7.2-mysql

$ apt install php-mongo

$ nano /etc/php/7.2/apache2/php-ini

extension=mongodb.so

Aquesta última comanda permet que el servidor reconegui el paquet instal·lat.

15.5 Configuració d'un servidor SFTP

SFTP és un servei que permet la pujada i descàrrega d'arxius d'una màquina servidor.

Per utilitzar un servidor SFTP en la màquina de l'administrador caldrà instal·lar els següents paquets de sistema:

$ apt get install openssh-client
$ apt get install openssh-server
$ apt get install openssh-sftp-server

Primerament s'escollirà en la màquina de l'administrador quin és el directori que es vol fer públic. Seguidament es recomana utilitzar un usuari que no tingui permisos d'administrador però que compti amb permisos de lectura i escriptura sobre aquest directori. Per concedir aquests permisos executarem les següents comandes:

$ mkdir /nom_directori
$ chmod 755 -R /nom_directori_remot
$ chown -R usuari /nom_directori_demot

Per configurar el servidor SFTP haurem de **configurar el servei ssh** per fer visible a la xarxa un directori remot. Per fer això modificarem el servei ssh modificant el fitxer de configuració sshd_config que es troba en el directori /etc/ssh:

$ nano /etc/ssh/sshd_config

Subsystem sftp /directori_remot
Match User nom_usuari_màquina
ForceCommand internal-sftp

Indicarem en el paràmetre **subsystem** el camí complet del directori remot on els agents pujaran els seus arxius. En Match User indicarem el nom de l'usuari que pot accedir mitjançant ssh a la màquina administrador. En Force-Command indicarem internal-sftp volen dir que el servidor es propi de la màquina i no és el que te ssh per

defecte. Per aplicar els canvis realitzats executarem la comanda:

$ service ssh restart

Solament podran enviar informació els agents que s'hagin connectat almenys una vegada al servidor SFTP com es comenta en l'apartat 9.1.4. Enviament de dades.

L'usuari, contrasenya, host i directori remot han de coincidir amb els que s'indiquen en el programa **upload.py** i el mòdul **service_execution**.